2.3 Machine-Independent Assembler Features

Literals, Symbol-Defining Statements, Expressions, Program Blocks, Control Sections and Program Linking

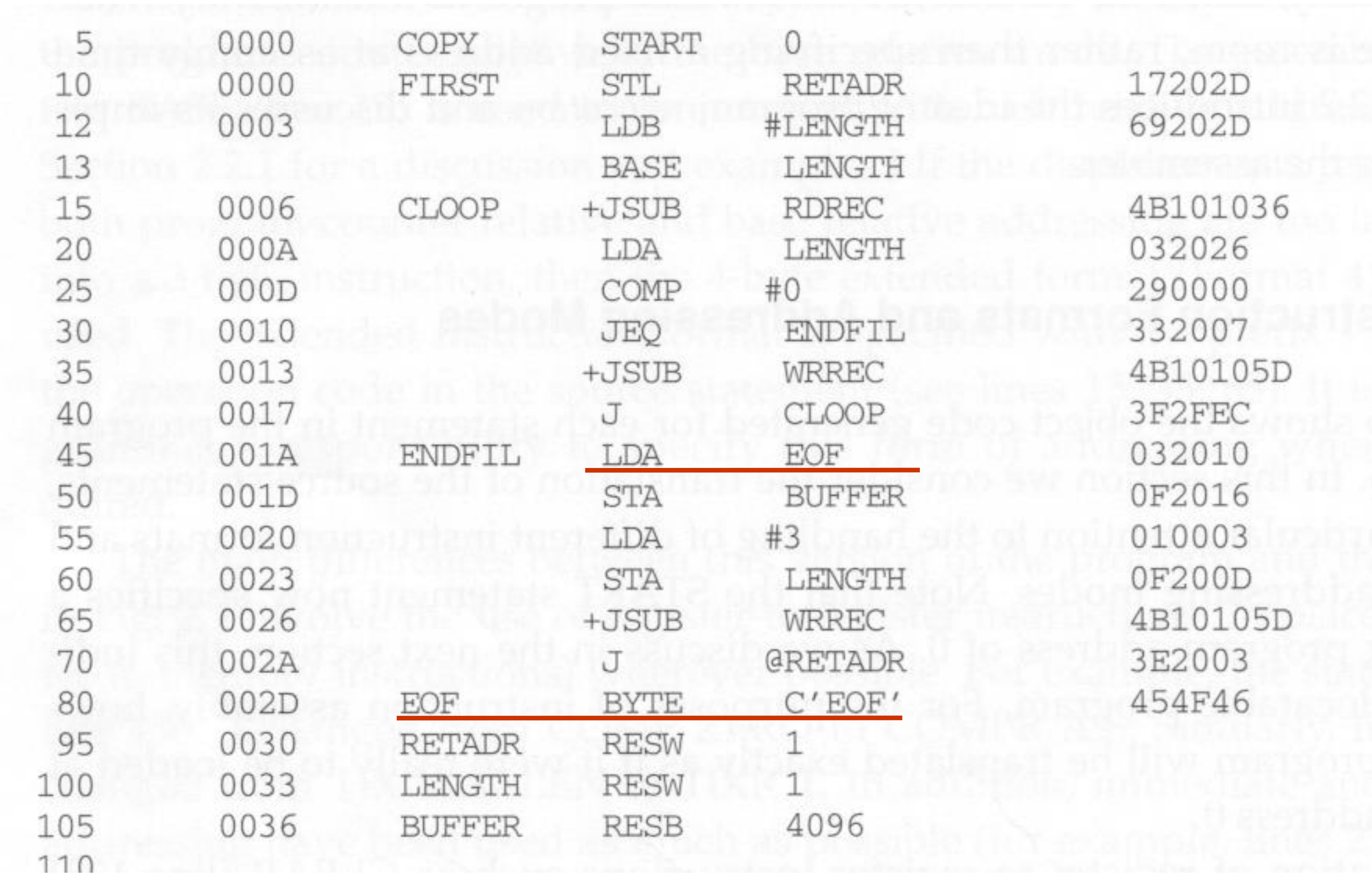
Machine Independent Assembler Features

- More related to issues about:
  - Programmer convenience
  - Software environment
- Common examples:
  - Literals
  - Symbol-defining statements
  - Expressions
  - Program blocks
  - Control sections
- Assembler directives are widely used to support these features

Literals

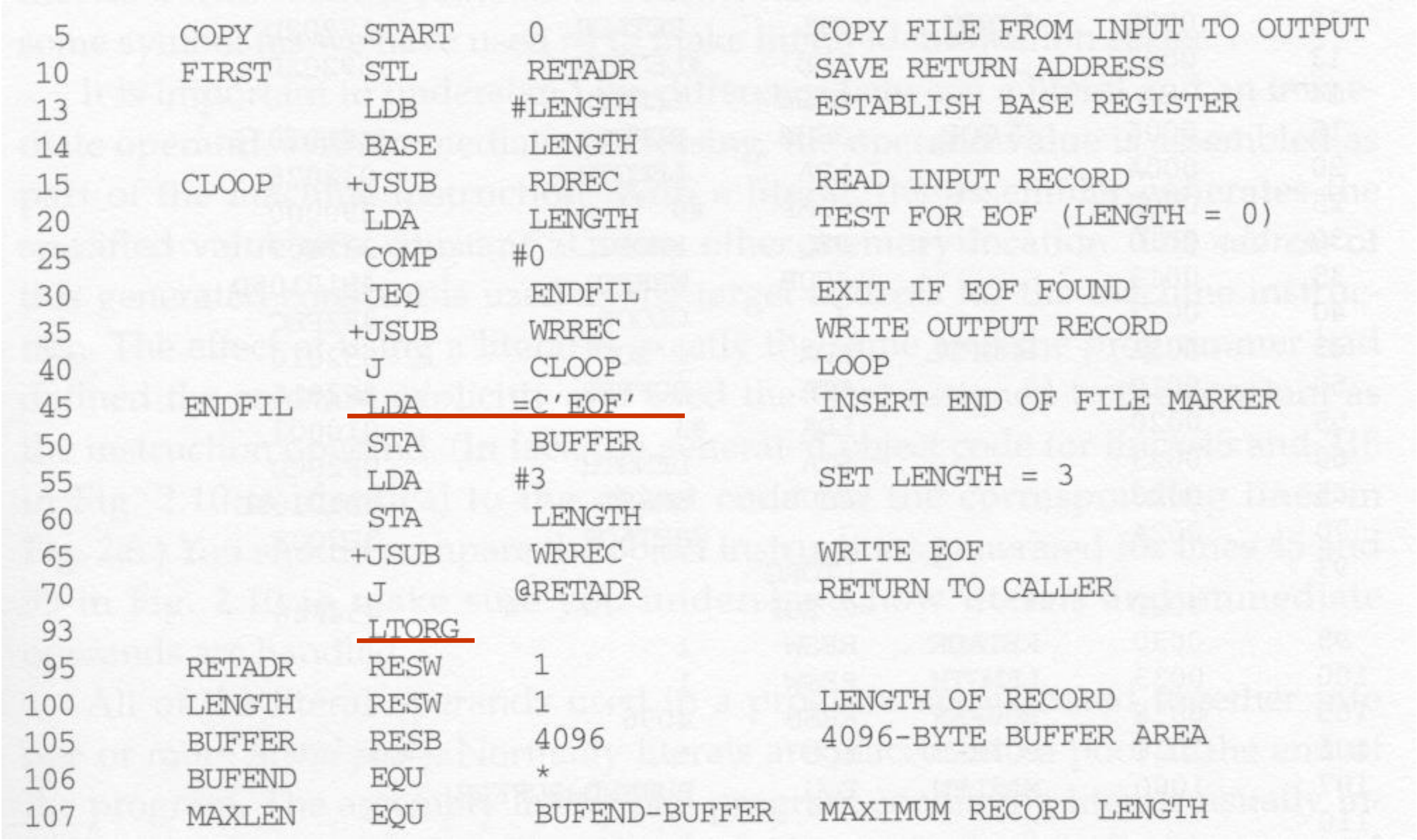
- Literal is equivalent to:
  - Define a constant explicitly and assign an address label for it
  - Use the label as the instruction operand
- Why use literals:
  - To avoid defining the constant somewhere and making up a label for it
  - Instead, to write the value of a constant operand as a part of the instruction
- How to use literals:
  - A literal is identified with the prefix =, followed by a specification of the literal value

Original Program



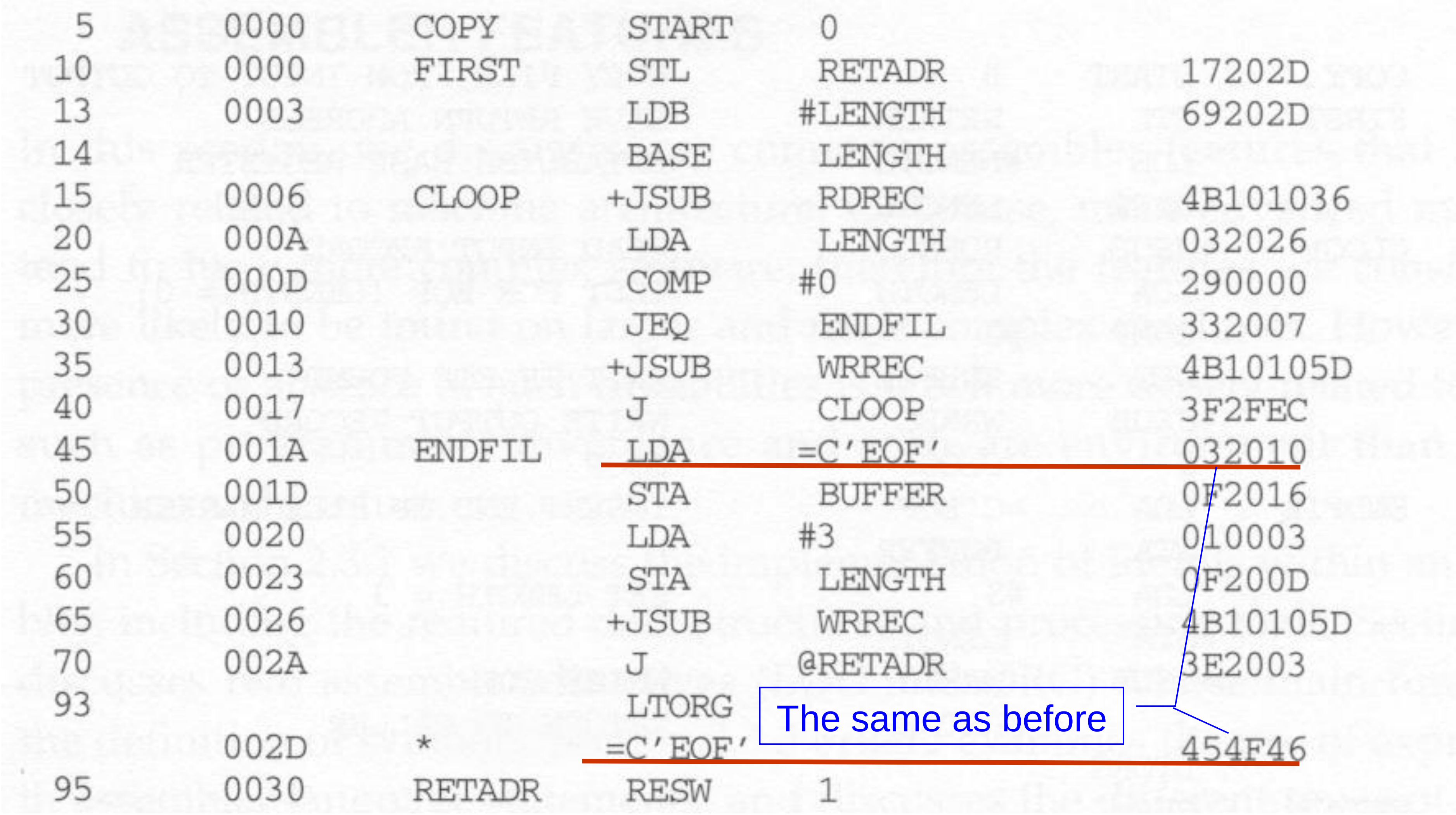
```
  5    0000    COPY      START    0
 10    0000    FIRST     STL      RETADR         17202D
 12    0003              LDB     #LENGTH         69202D
 13                      BASE     LENGTH
 15    0006    CLOOP    +JSUB     RDREC          4B101036
 20    000A              LDA      LENGTH         032026
 25    000D              COMP    #0              290000
 30    0010              JEQ      ENDFIL         332007
 35    0013             +JSUB     WRREC          4B10105D
 40    0017              J        CLOOP          3F2FEC
 45    001A    ENDFIL    LDA      EOF            032010
 50    001D              STA      BUFFER         0F2016
 55    0020              LDA     #3              010003
 60    0023              STA      LENGTH         0F200D
 65    0026             +JSUB     WRREC          4B10105D
 70    002A              J       @RETADR         3E2003
 80    002D    EOF       BYTE     C'EOF'         454F46
 95    0030    RETADR    RESW     1
100    0033    LENGTH    RESW     1
105    0036    BUFFER    RESB     4096
110
```

Using Literal



```
  5   COPY     START   0              COPY FILE FROM INPUT TO OUTPUT
 10   FIRST    STL     RETADR         SAVE RETURN ADDRESS
 13            LDB     #LENGTH        ESTABLISH BASE REGISTER
 14            BASE    LENGTH
 15   CLOOP   +JSUB    RDREC          READ INPUT RECORD
 20            LDA     LENGTH         TEST FOR EOF (LENGTH = 0)
 25            COMP    #0
 30            JEQ     ENDFIL         EXIT IF EOF FOUND
 35           +JSUB    WRREC          WRITE OUTPUT RECORD
 40            J       CLOOP          LOOP
 45   ENDFIL   LDA     =C'EOF'        INSERT END OF FILE MARKER
 50            STA     BUFFER
 55            LDA     #3             SET LENGTH = 3
 60            STA     LENGTH
 65           +JSUB    WRREC          WRITE EOF
 70            J       @RETADR        RETURN TO CALLER
 93            LTORG
 95   RETADR   RESW    1
100   LENGTH   RESW    1              LENGTH OF RECORD
105   BUFFER   RESB    4096           4096-BYTE BUFFER AREA
106   BUFEND   EQU     *
107   MAXLEN   EQU     BUFEND-BUFFER  MAXIMUM RECORD LENGTH
```

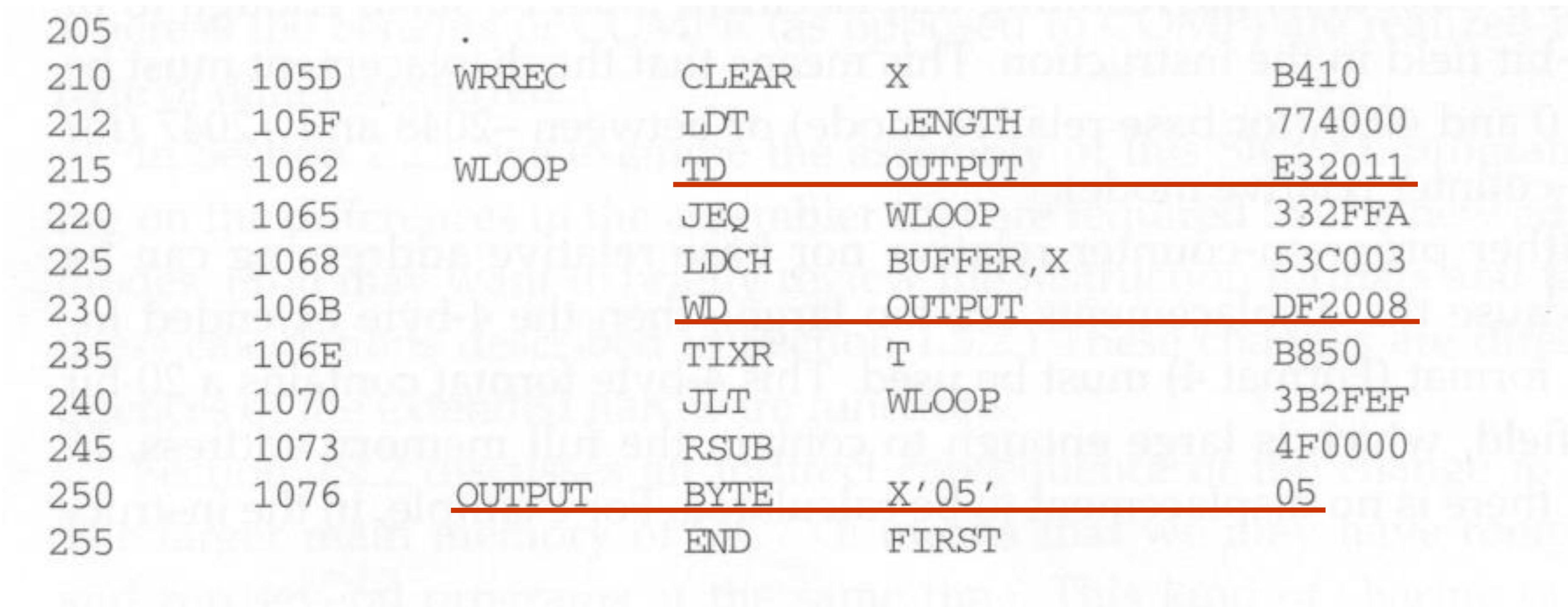
Object Program Using Literal



```
  5    0000    COPY      START    0
 10    0000    FIRST     STL      RETADR          17202D
 13    0003              LDB     #LENGTH          69202D
 14                      BASE     LENGTH
 15    0006    CLOOP    +JSUB     RDREC           4B101036
 20    000A              LDA      LENGTH          032026
 25    000D              COMP    #0               290000
 30    0010              JEQ      ENDFIL          332007
 35    0013             +JSUB     WRREC           4B10105D
 40    0017              J        CLOOP           3F2FEC
 45    001A    ENDFIL    LDA     =C'EOF'          032010
 50    001D              STA      BUFFER          0F2016
 55    0020              LDA     #3               010003
 60    0023              STA      LENGTH          0F200D
 65    0026             +JSUB     WRREC           4B10105D
 70    002A              J       @RETADR          3E2003
 93                      LTORG
       002D    *        =C'EOF'                   454F46
 95    0030    RETADR    RESW     1
```

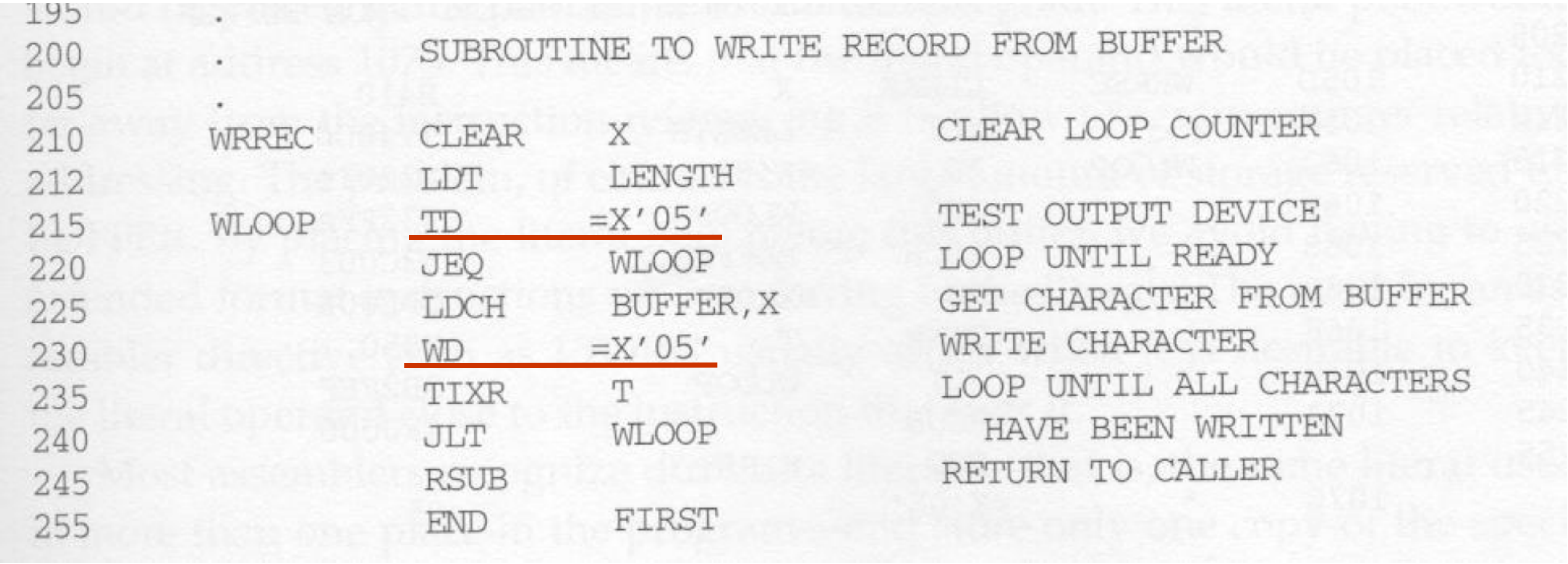
The same as before

Original Program



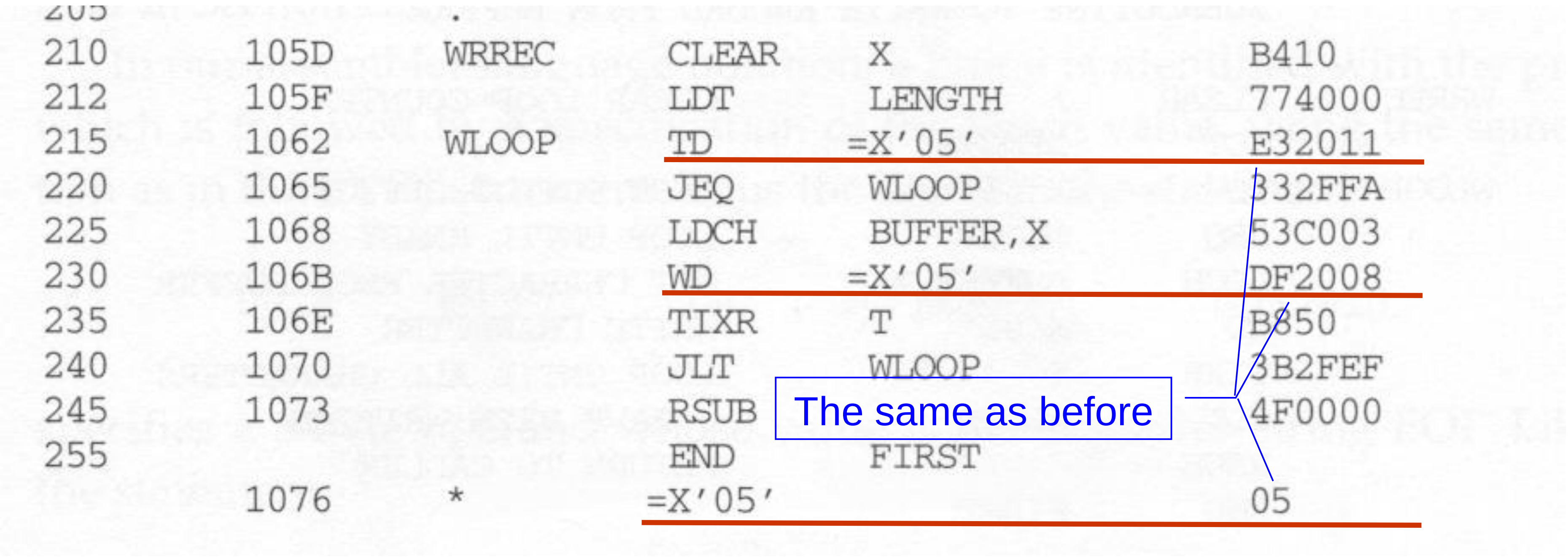
```
205                 .
210     105D        WRREC     CLEAR     X                B410
212     105F                  LDT       LENGTH           774000
215     1062        WLOOP     TD        OUTPUT           E32011
220     1065                  JEQ       WLOOP            332FFA
225     1068                  LDCH      BUFFER,X         53C003
230     106B                  WD        OUTPUT           DF2008
235     106E                  TIXR      T                B850
240     1070                  JLT       WLOOP            3B2FEF
245     1073                  RSUB                       4F0000
250     1076        OUTPUT    BYTE      X'05'            05
255                           END       FIRST
```

Using Literal



```
.
.          SUBROUTINE TO WRITE RECORD FROM BUFFER
.
WRREC      CLEAR      X                  CLEAR LOOP COUNTER
           LDT        LENGTH
WLOOP      TD        =X'05'              TEST OUTPUT DEVICE
           JEQ        WLOOP              LOOP UNTIL READY
           LDCH       BUFFER,X           GET CHARACTER FROM BUFFER
           WD        =X'05'              WRITE CHARACTER
           TIXR       T                  LOOP UNTIL ALL CHARACTERS
           JLT        WLOOP                HAVE BEEN WRITTEN
           RSUB                          RETURN TO CALLER
           END        FIRST
```

Object Program Using Literal



```
205             .
210     105D    WRREC   CLEAR   X               B410
212     105F            LDT     LENGTH          774000
215     1062    WLOOP   TD     =X'05'           E32011
220     1065            JEQ     WLOOP           332FFA
225     1068            LDCH    BUFFER,X        53C003
230     106B            WD     =X'05'           DF2008
235     106E            TIXR    T               B850
240     1070            JLT     WLOOP           3B2FEF
245     1073            RSUB                    4F0000
255                     END     FIRST
        1076    *      =X'05'                   05
```

The same as before

Literal vs. Immediate Addressing

- Same:
  - Operand field contains constant values in source code
- Difference:
  - Immediate addressing: the assembler put the constant value as part of the machine instruction
  - Literal: the assembler store the constant value elsewhere and put that address as part of the machine instruction

Literal Pool

- All of the literal operands are gathered together into one or more literal pools.
- Where is the literal pool:
  - At the end of the object program, generated immediately following the END statement
  - At the location where the LTORG directive is encountered
    - To keep the literal operand close to the instruction that uses it

Duplicate Literals

- Duplicate literals:
  - The same literal used more than once in the program
  - Only one copy of the specified value needs to be stored
  - For example, =X’05’ in the example program
- How to recognize the duplicate literals
  - Compare the character strings defining them
    - Easier to implement, but has potential problem (see next)
    - E.g., =X’05’
  - Compare the generated data value
    - Better, but will increase the complexity of the assembler
    - E.g., =C’EOF’ and =X’454F46’

Problem of Duplicate-Literal Recognition using Character Strings

- There may be some literals that have the same name, but different values
- For example, the literal whose value depends on its location in the program
  - The value of location counter denoted by *

    ```
    BASE      *
    LDB      =*
    ```

  - The literal =* repeatedly used in the program has the same name, but different values
- All this kind of literals have to be stored in the literal pool

Implementation of Literal

- Data structure: a literal table LITTAB
  - Literal name
  - Operand value and length
  - Address
- LITTAB is often organized as a hash table, using the literal name or value as the key

Implementation of Literal

- Pass 1
  - As each literal operand is recognized
    - Search the LITTAB for the specified literal name or value
    - If the literal is already present, no action is needed
    - Otherwise, the literal is added to LITTAB (store the name, value, and length, but not address)
  - As LTORG or END is encountered
    - Scan the LITTAB
    - For each literal with empty address field, assign the address and update the LOCCTR accordingly

Implementation of Literal

- Pass 2
  - As each literal operand is recognized
    - Search the LITTAB for the specified literal name or value
    - If the literal is found, use the associated address as the operand of the instruction
    - Otherwise, error (should not happen)
  - As LTORG or END is encountered
    - insert the data values of the literals in the object program
  - Modification record is generated if necessary

Symbol-Defining Statements

- How to define symbols and their values
  - Address label
    - The label is the symbol name and the assigned address is its value

```
FIRST     STL     RETADR
```

  - Assembler directive EQU

```
symbol    EQU    value
```

    - This statement enters the symbol into SYMTAB and assigns to it the value specified
    - The value can be a constant or an expression
  - Assembler directive ORG

```
          ORG    value
```

    - The assembler reset its LOCCTR to the specified value

Use of EQU

- To improve the program readability, avoid using the magic numbers, make it easier to find and change constant values
  - Replace

    ```
    +LDT   #4096
    ```

  - with

    ```
    MAXLEN   EQU   4096
    +LDT   #MAXLEN
    ```

- To define mnemonic names for registers
  - A EQU   0
  - X EQU   1
  - BASE   EQU R1
  - COUNT EQU R2

Use of ORG

- Indirect value assignment:

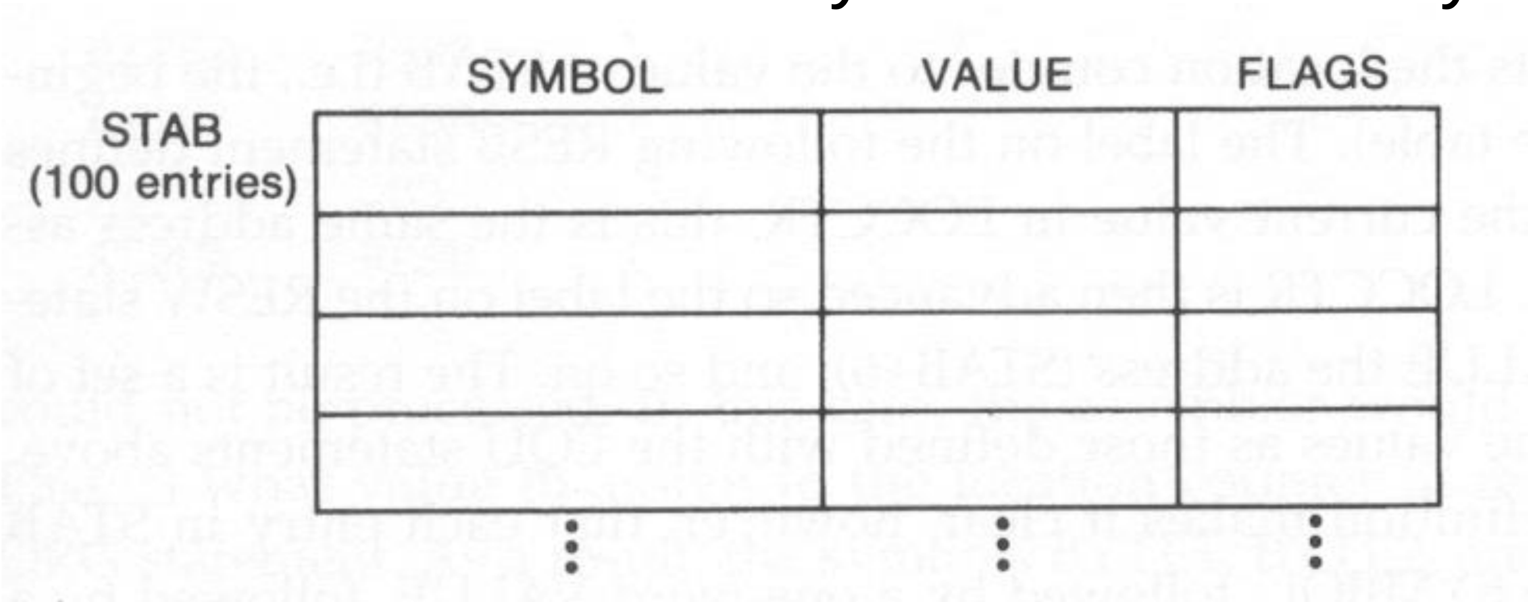
  ORG value

  - When ORG is encountered, the assembler resets its LOCCTR to the specified value
  - ORG will affect the values of all labels defined until the next ORG
  - If the previous value of LOCCTR can be automatically remembered, we can return to the normal use of LOCCTR by simply write

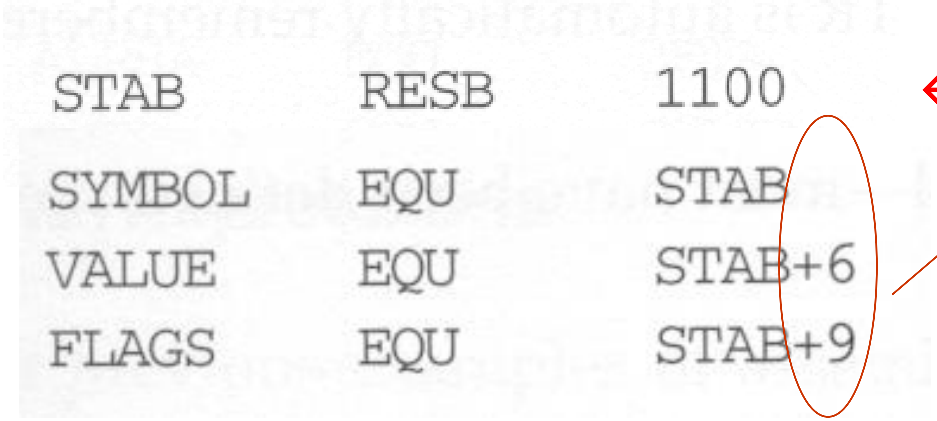
    ORG

Example of Using ORG

- Consider the following data structure (11 bytes)
  - – SYMBOL: 6 bytes
  - – VALUE: 3 bytes (one word)
  - – FLAGS: 2 bytes
- we want to refer to every field of each entry



Not Using ORG



```
STAB       RESB     1100
SYMBOL     EQU      STAB
VALUE      EQU      STAB+6
FLAGS      EQU      STAB+9
```

←(100 entries x 11bytes)

Offsets from STAB
Less readable and
meaningful

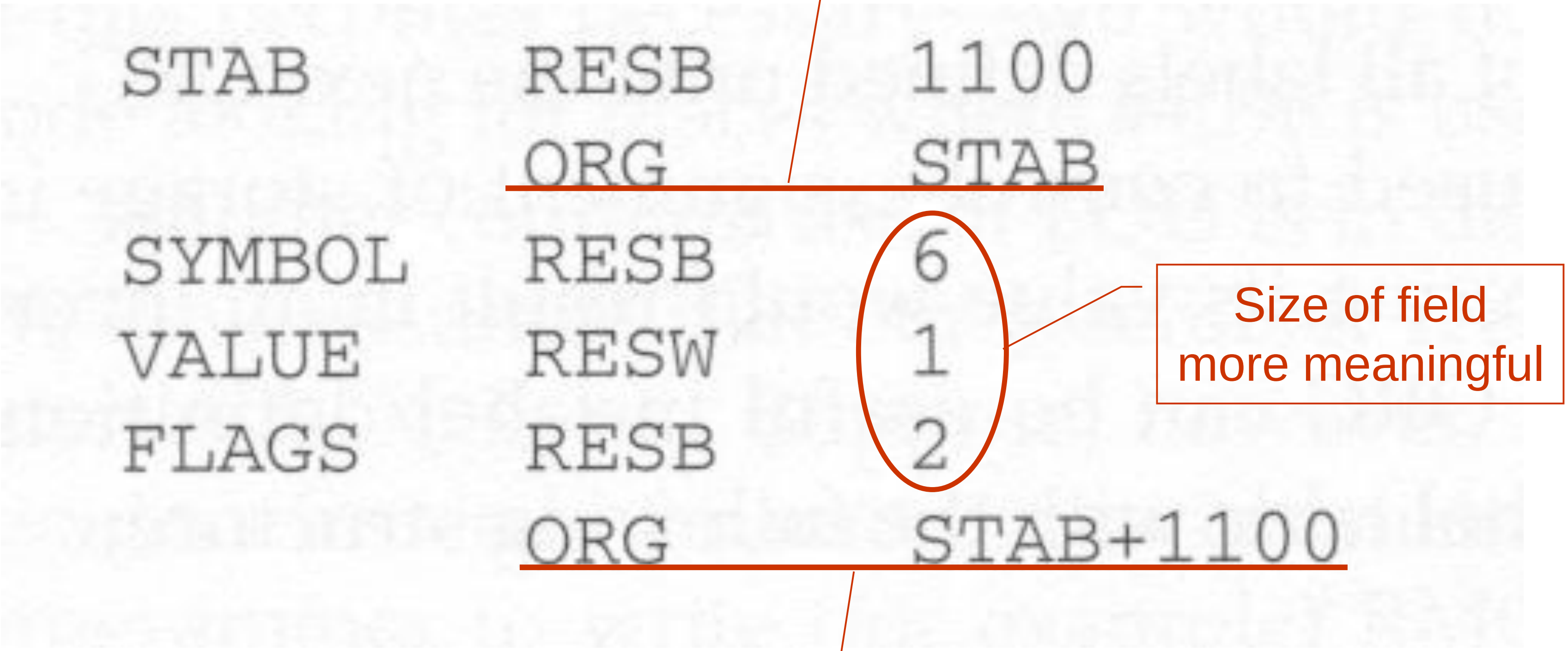
- We can fetch the VALUE field by

  LDA VALUE,X

- X = 0, 11, 22, … for each entry

Using ORG

Set the LOCCTR to STAB



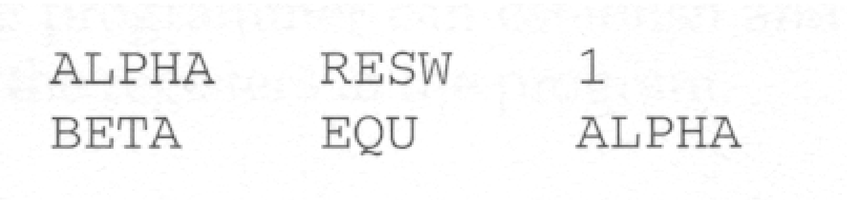
```
STAB      RESB    1100
          ORG     STAB
SYMBOL    RESB    6
VALUE     RESW    1
FLAGS     RESB    2
          ORG     STAB+1100
```

Size of field more meaningful

Restore the LOCCTR to its previous value

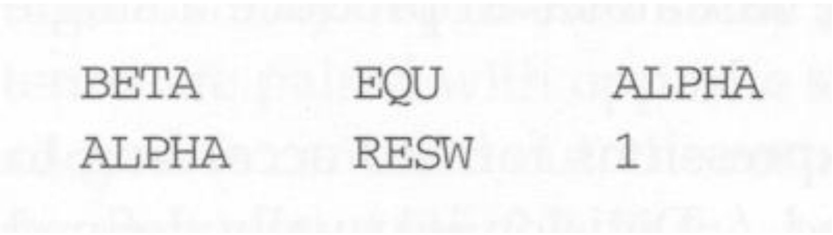
Forward-Reference Problem

- Forward reference is not allowed here for EQU and ORG.
- That is, all terms in the value field must have been defined previously in the program.
- The reason is that all symbols must have been defined during Pass 1 in a two-pass assembler.



```
ALPHA      RESW      1
BETA       EQU       ALPHA
```

Allowed



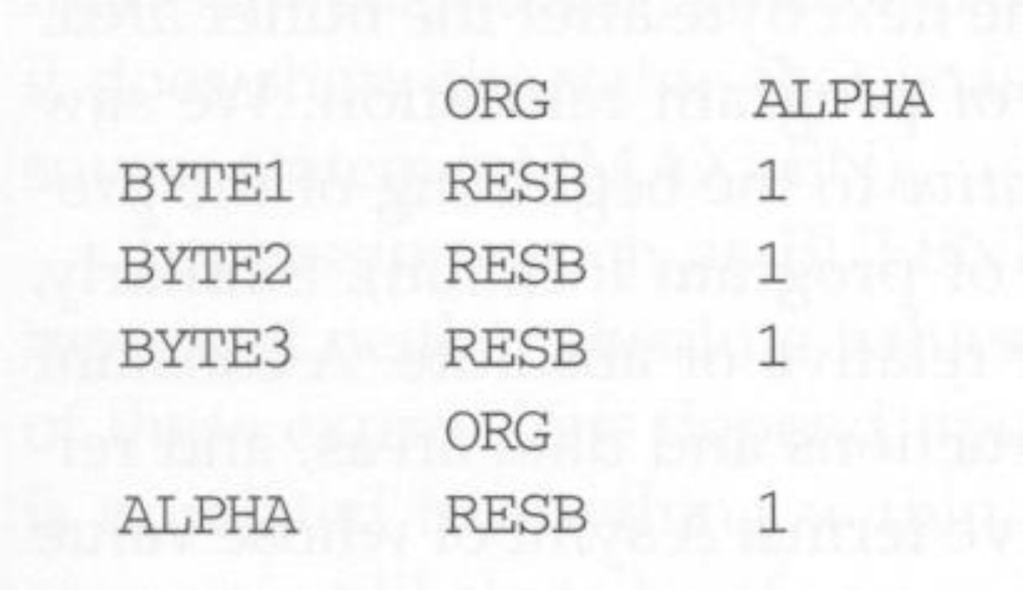
```
BETA       EQU       ALPHA
ALPHA      RESW      1
```

Not allowed

Forward-Reference Problem

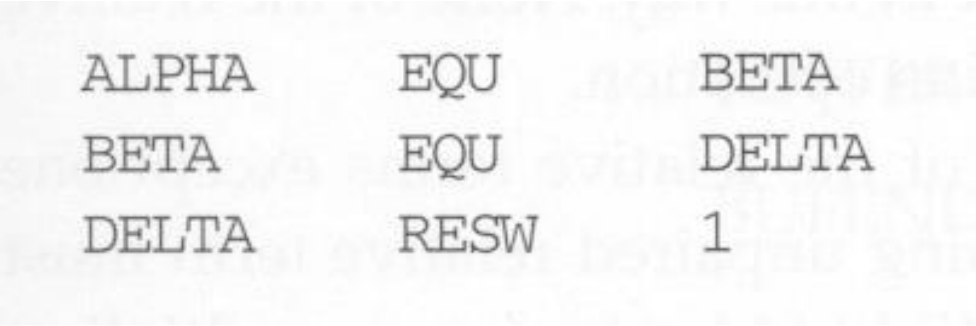
- ALPHA EQU BETA
- BETA EQU GAMMA
- GAMMA EQU 1

Forward-Reference Problem



```
          ORG       ALPHA
BYTE1     RESB      1
BYTE2     RESB      1
BYTE3     RESB      1
          ORG
ALPHA     RESB      1
```

Not allowed



```
ALPHA     EQU       BETA
BETA      EQU       DELTA
DELTA     RESW      1
```

Not allowed

Expressions

- A single term as an instruction operand can be replaced by an expression.

```
STAB    RESB    1100
                  ⇩
STAB    RESB    11*100
                  ⇩
STAB    RESB    (6+3+2)*MAXENTRIES
```

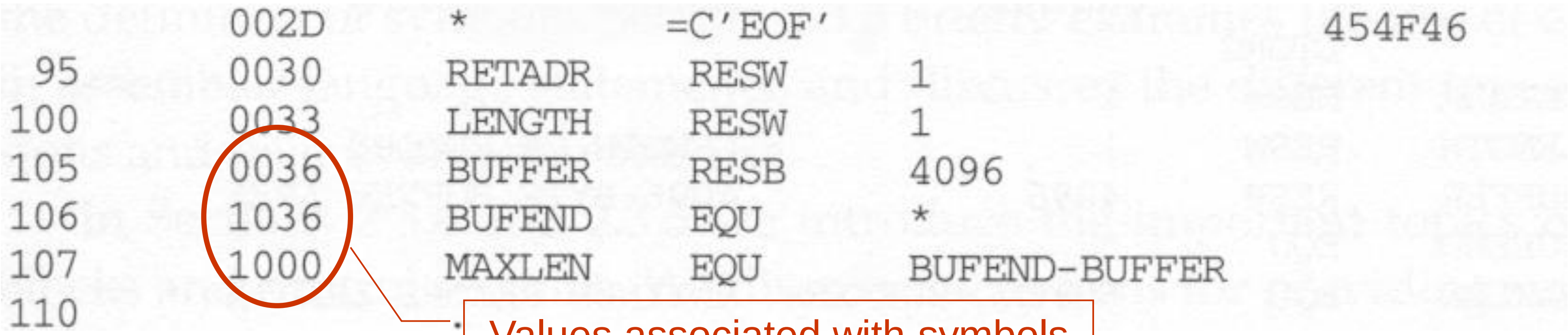
- The assembler has to evaluate the expression to produce a single operand address or value.
- Expressions consist of
  - Operator
    - +, -, *, / (division is usually defined to produce an integer result)
  - Individual terms
    - Constants
    - User-defined symbols
    - Special terms, e.g., *, the current value of LOCCTR

Relocation Problem in Expressions

- Values of terms can be
  - Absolute (independent of program location)
    - constants
  - Relative (to the beginning of the program)
    - Address labels
    - * (value of LOCCTR)
- Expressions can be
  - Absolute
    - Only absolute terms
    - Relative terms in pairs with opposite signs for each pair
  - Relative
    - All the relative terms except one can be paired as described in “absolute”. The remaining unpaired relative term must have a positive sign.
- No relative terms may enter into a multiplication or division operation
- Expressions that do not meet the conditions of either “absolute” or “relative” should be flagged as errors.

Absolute Expression

- Relative term or expression implicitly represents (S+r)
  - S: the starting address of the program
  - r: value of the term or expression relative to S
- For example
  - BUFFER: S+r1, BUFEND: S+r2
- The expression, BUFEND-BUFFER, is absolute.
  - MAXLEN = (S+r2)-(S+r1) = r2-r1   (no S here)
  - MAXLEN means the length of the buffer area
- Illegal expressions: BUFEND+BUFFER, 100-BUFFER, 3*BUFFER

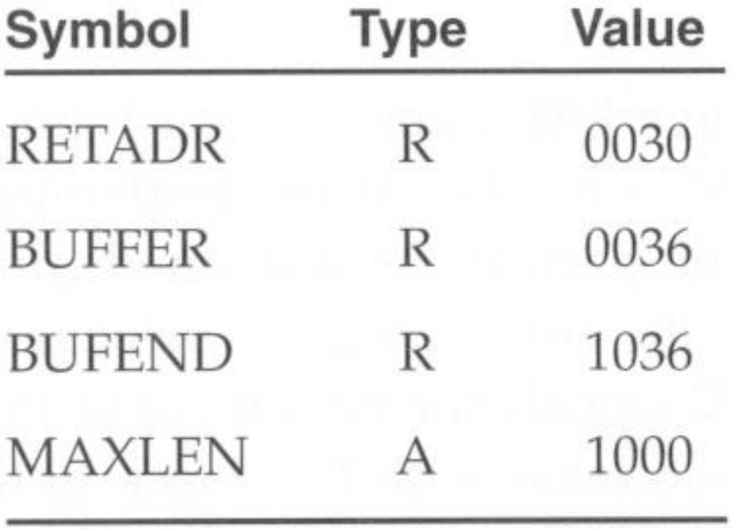


```
        002D     *          =C'EOF'                 454F46
 95     0030     RETADR     RESW      1
100     0033     LENGTH     RESW      1
105     0036     BUFFER     RESB      4096
106     1036     BUFEND     EQU       *
107     1000     MAXLEN     EQU       BUFEND-BUFFER
110                .
```

Values associated with symbols

Absolute or Relative

- To determine the type of an expression, we must keep track of the types of all symbols defined in the program.
- We need a “flag” in the SYMTAB for indication.



Symbol	Type	Value
RETADR	R	0030
BUFFER	R	0036
BUFEND	R	1036
MAXLEN	A	1000

Program Blocks and Control Sections

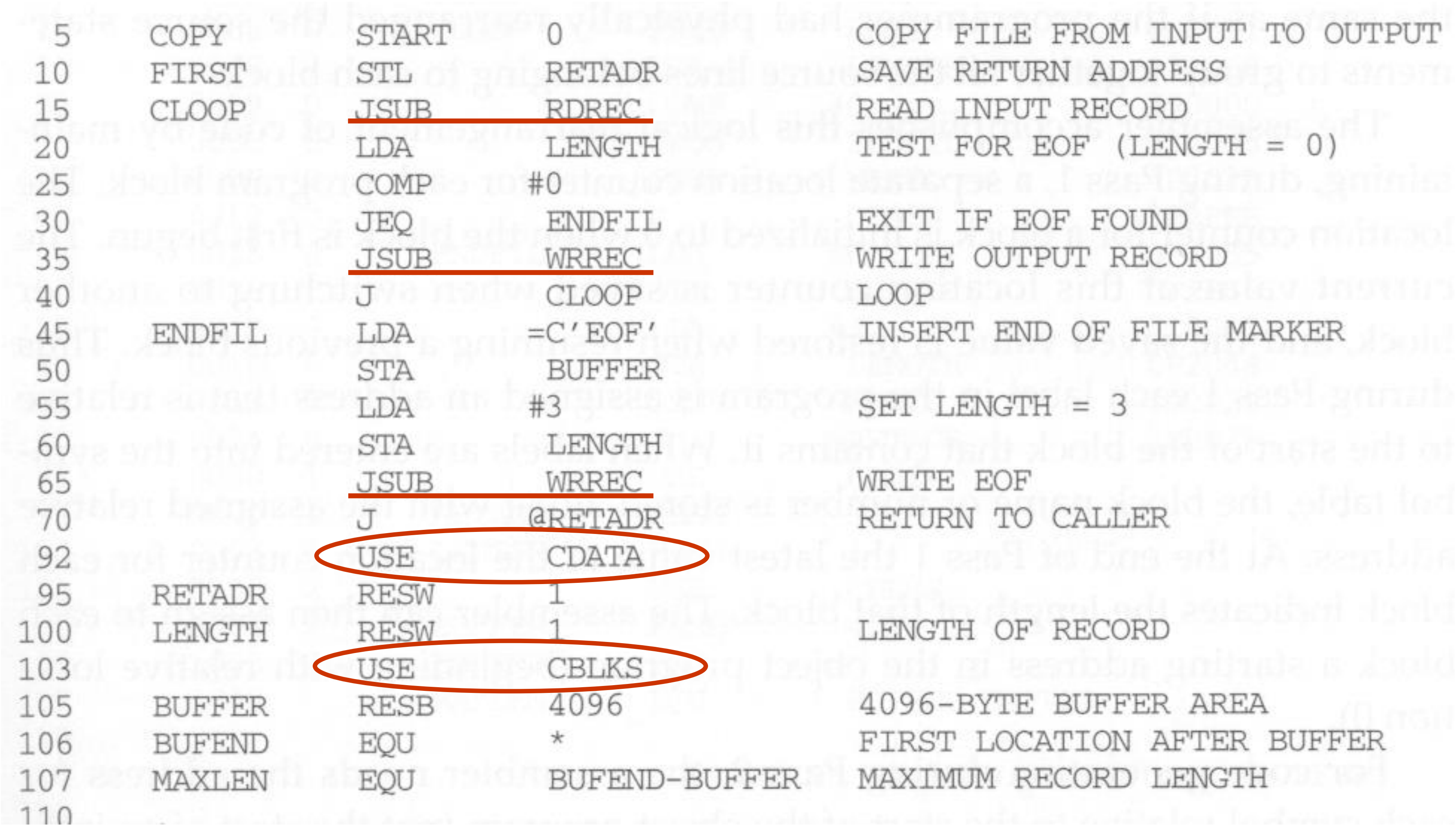
- Although the source program logically contains subroutines, data area, etc, they were assembled into a single block of object code in which the machine instructions and data appeared in the same order as they were in the source program.
- To provide flexibility:
  - Program blocks
    - Segments of code that are rearranged within a single object program unit
  - Control sections
    - Segments of code that are translated into independent object program units

Program Blocks

- As an *example*, three blocks are used:
  - default: executable instructions
  - CDATA: all data areas that are less in length
  - CBLKS: all data areas that consists of larger blocks of memory
- The assembler directive USE indicates which portions of the source program belong to the various blocks.

Program with Multiple Program Blocks

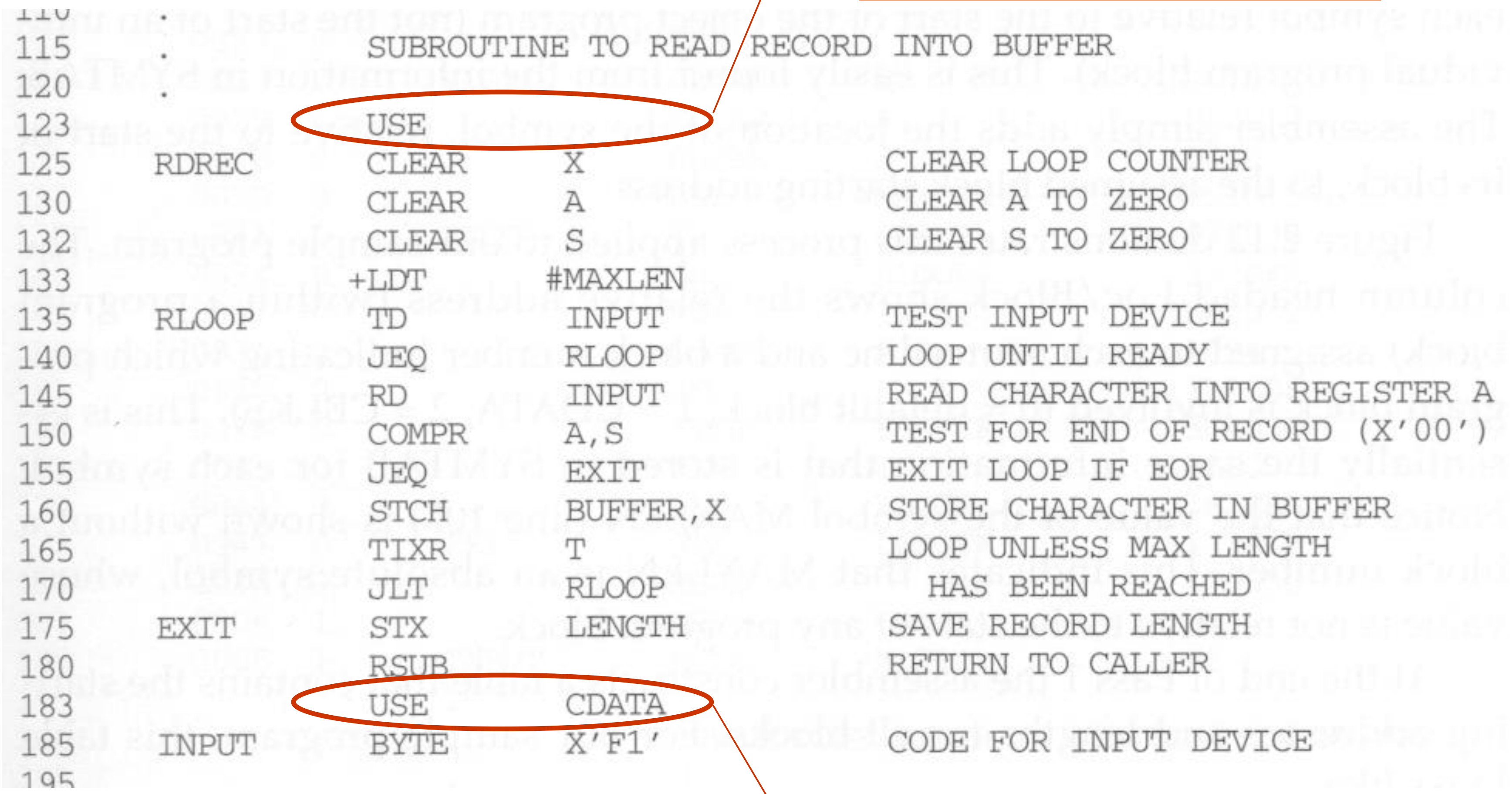
At the beginning, the default block is assumed.



```
  5     COPY      START    0               COPY FILE FROM INPUT TO OUTPUT
 10     FIRST     STL      RETADR          SAVE RETURN ADDRESS
 15     CLOOP     JSUB     RDREC           READ INPUT RECORD
 20               LDA      LENGTH          TEST FOR EOF (LENGTH = 0)
 25               COMP     #0
 30               JEQ      ENDFIL          EXIT IF EOF FOUND
 35               JSUB     WRREC           WRITE OUTPUT RECORD
 40               J        CLOOP           LOOP
 45     ENDFIL    LDA      =C'EOF'         INSERT END OF FILE MARKER
 50               STA      BUFFER
 55               LDA      #3              SET LENGTH = 3
 60               STA      LENGTH
 65               JSUB     WRREC           WRITE EOF
 70               J        @RETADR         RETURN TO CALLER
 92               USE      CDATA
 95     RETADR    RESW     1
100     LENGTH    RESW     1               LENGTH OF RECORD
103               USE      CBLKS
105     BUFFER    RESB     4096            4096-BYTE BUFFER AREA
106     BUFEND    EQU      *               FIRST LOCATION AFTER BUFFER
107     MAXLEN    EQU      BUFEND-BUFFER   MAXIMUM RECORD LENGTH
110
```

Program with Multiple Program Blocks

Resume the default block

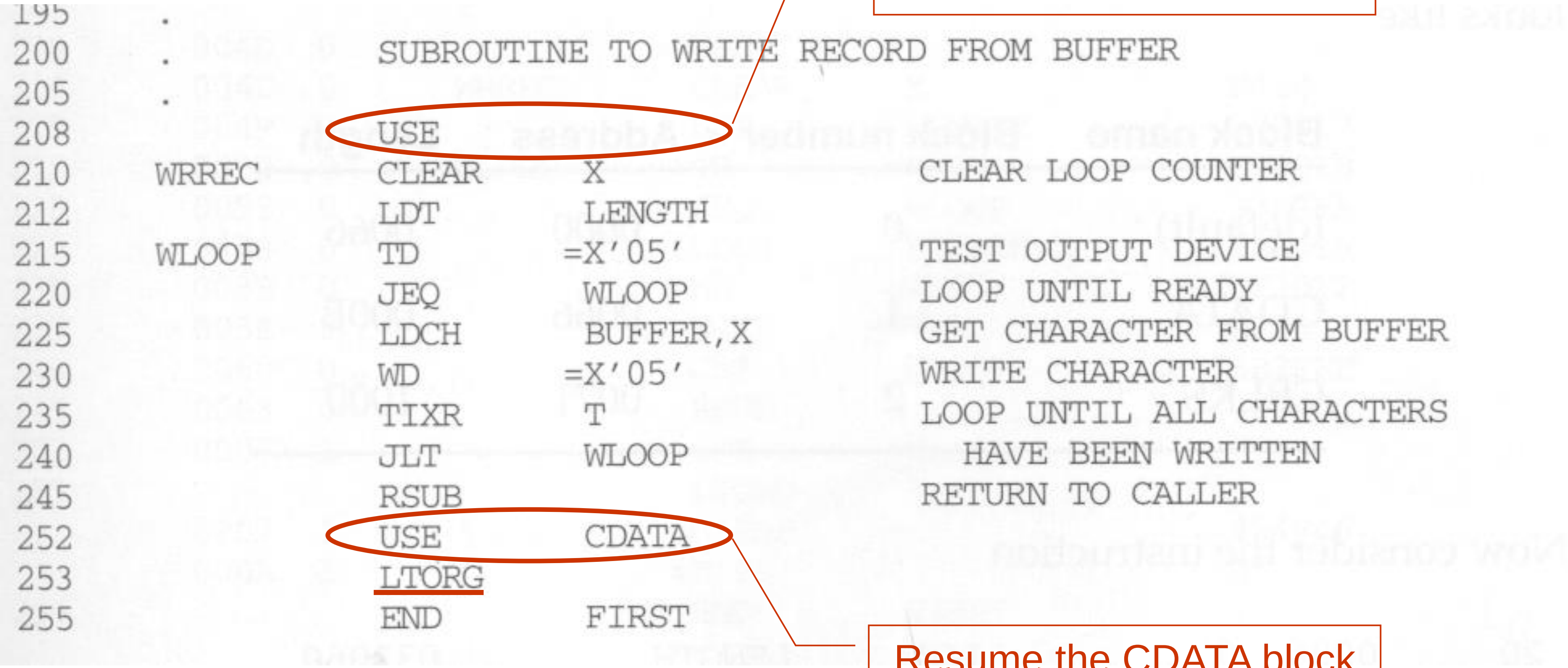


```
110     .
115     .          SUBROUTINE TO READ RECORD INTO BUFFER
120     .
123                USE
125     RDREC      CLEAR     X                   CLEAR LOOP COUNTER
130                CLEAR     A                   CLEAR A TO ZERO
132                CLEAR     S                   CLEAR S TO ZERO
133               +LDT      #MAXLEN
135     RLOOP      TD        INPUT               TEST INPUT DEVICE
140                JEQ       RLOOP               LOOP UNTIL READY
145                RD        INPUT               READ CHARACTER INTO REGISTER A
150                COMPR     A,S                 TEST FOR END OF RECORD (X'00')
155                JEQ       EXIT                EXIT LOOP IF EOR
160                STCH      BUFFER,X            STORE CHARACTER IN BUFFER
165                TIXR      T                   LOOP UNLESS MAX LENGTH
170                JLT       RLOOP                 HAS BEEN REACHED
175     EXIT       STX       LENGTH              SAVE RECORD LENGTH
180                RSUB                          RETURN TO CALLER
183                USE       CDATA
185     INPUT      BYTE      X'F1'               CODE FOR INPUT DEVICE
195
```

Resume the CDATA block

Program with Multiple Program Blocks

Resume the default block



```
195     .
200     .          SUBROUTINE TO WRITE RECORD FROM BUFFER
205     .
208                USE
210     WRREC      CLEAR      X                CLEAR LOOP COUNTER
212                LDT        LENGTH
215     WLOOP      TD         =X’05’           TEST OUTPUT DEVICE
220                JEQ        WLOOP            LOOP UNTIL READY
225                LDCH       BUFFER,X         GET CHARACTER FROM BUFFER
230                WD         =X’05’           WRITE CHARACTER
235                TIXR       T                LOOP UNTIL ALL CHARACTERS
240                JLT        WLOOP              HAVE BEEN WRITTEN
245                RSUB                        RETURN TO CALLER
252                USE        CDATA
253                LTORG
255                END        FIRST
```

Resume the CDATA block

Program Blocks

- Each program block may actually contain several separate segments of the source program.
- The assembler will logically rearrange these segments to gather together the pieces of each block.
- The result is the same as if the programmer had physically rearranged the source statements to group together all the source lines belonging to each block.

Why Program Blocks

- To satisfy the contradictive goals:
  - Separate the program into blocks in a particular order
    - Large buffer area is moved to the end of the object program
    - Using the extended format instructions or base relative mode may be reduced. (lines 15, 35, and 65)
    - Placement of literal pool is easier: simply put them before the large data area, CDATA block. (line 253)
  - Data areas are scattered
    - Program readability is better if data areas are placed in the source program close to the statements that reference them.

How to Rearrange Codes into Program Blocks

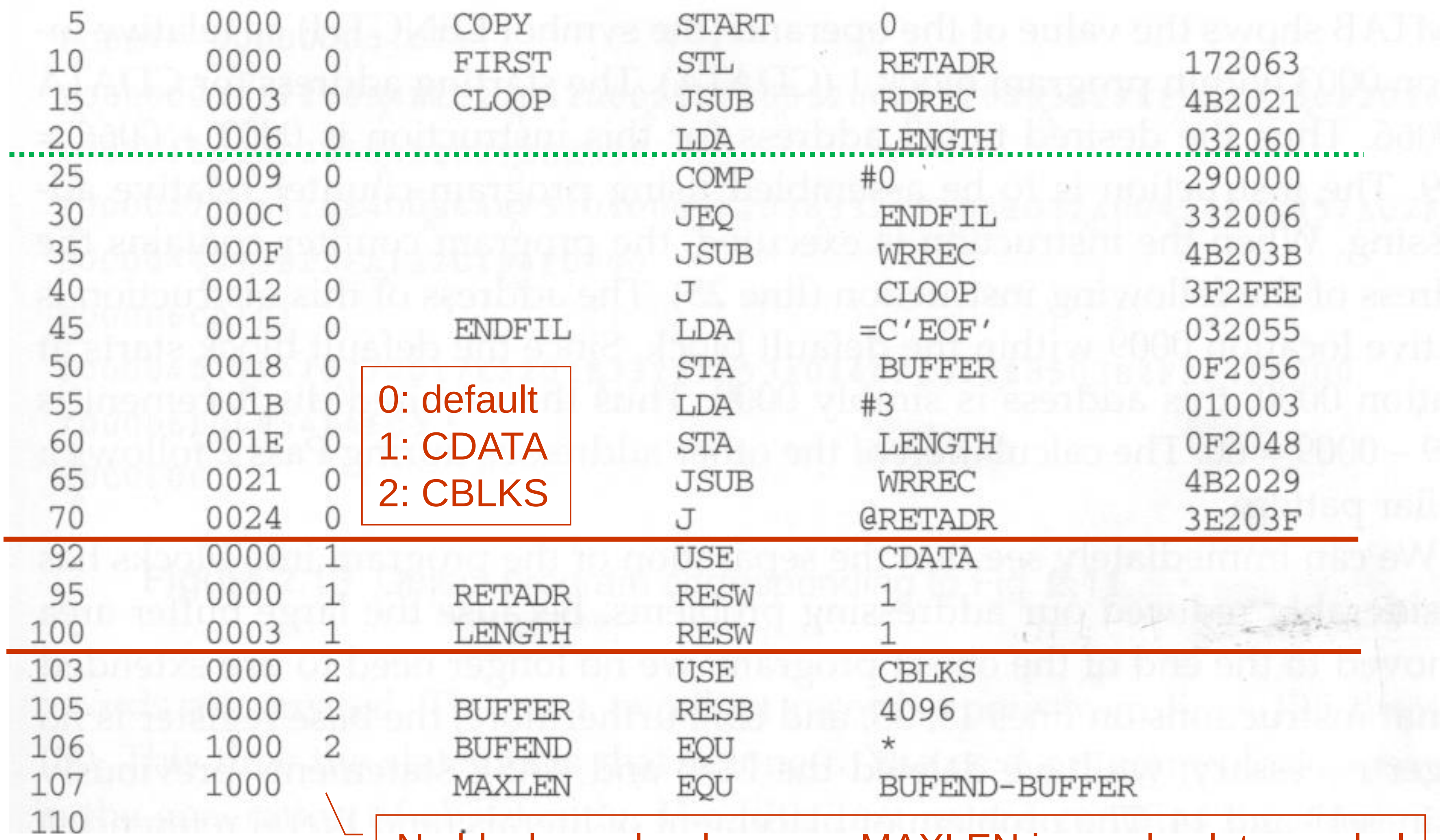
- Pass 1
  - Maintain a separate LOCCTR for each program block
    - initialized to 0 when the block is first begun
    - saved when switching to another block
    - restored when resuming a previous block
  - Assign to each label an address relative to the start of the block that contains it
  - Store the block name or number in the SYMTAB along with the assigned relative address of the label
  - Indicate the block length as the latest value of LOCCTR for each block at the end of Pass1
  - Assign to each block a starting address in the object program by concatenating the program blocks in a particular order

How to Rearrange Codes into Program Blocks

- Pass 2
  - Calculate the address for each symbol relative to the start of the object program by adding
    - the location of the symbol relative to the start of its block
    - the assigned starting address of this block

Object Program with Multiple Program Blocks

Loc/Block

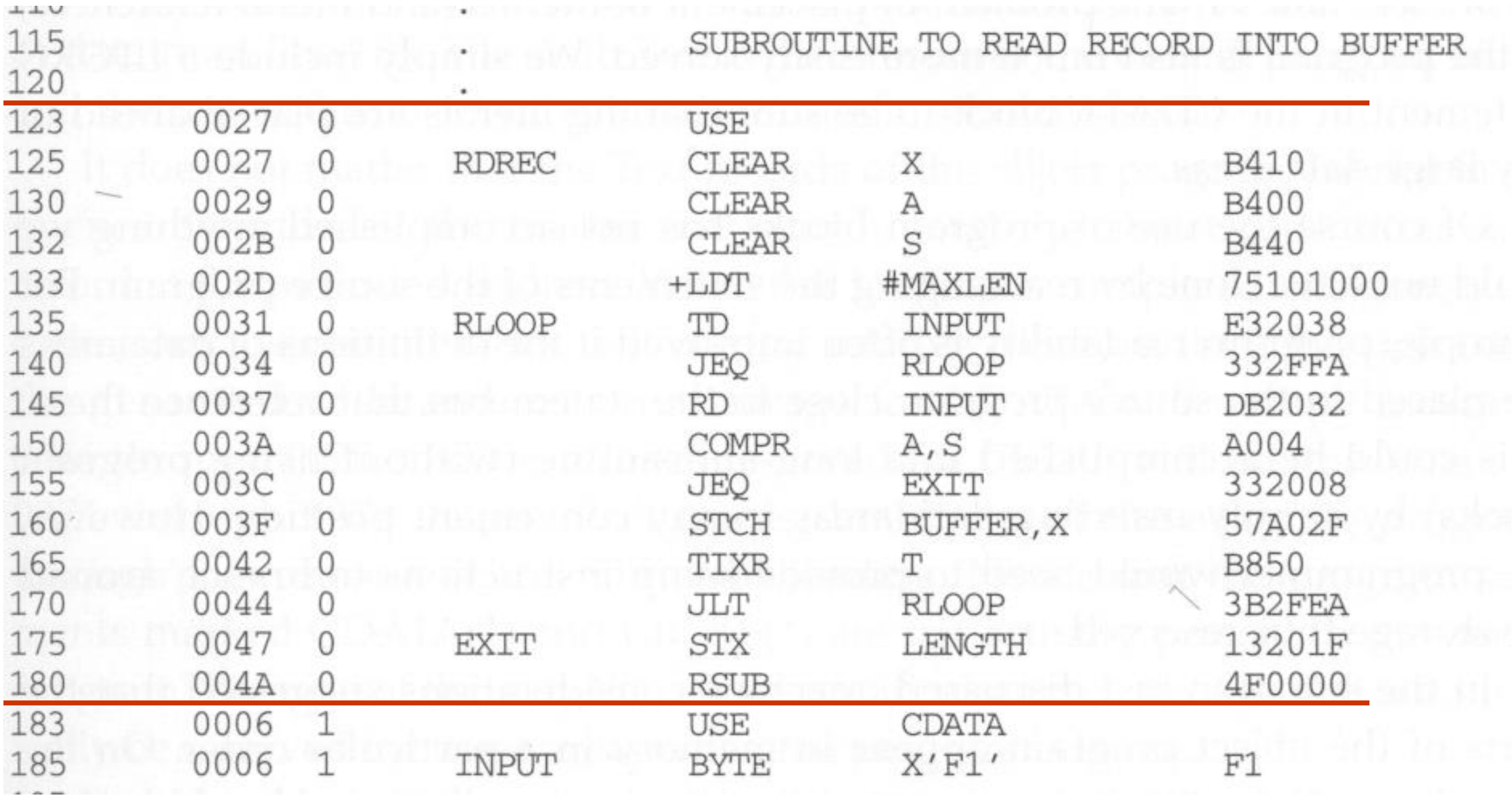


```
  5   0000 0   COPY     START   0
 10   0000 0   FIRST    STL     RETADR          172063
 15   0003 0   CLOOP    JSUB    RDREC           4B2021
 20   0006 0            LDA     LENGTH          032060
 25   0009 0            COMP   #0               290000
 30   000C 0            JEQ     ENDFIL          332006
 35   000F 0            JSUB    WRREC           4B203B
 40   0012 0            J       CLOOP           3F2FEE
 45   0015 0   ENDFIL   LDA    =C'EOF'          032055
 50   0018 0            STA     BUFFER          0F2056
 55   001B 0            LDA    #3               010003
 60   001E 0            STA     LENGTH          0F2048
 65   0021 0            JSUB    WRREC           4B2029
 70   0024 0            J      @RETADR          3E203F
 92   0000 1            USE     CDATA
 95   0000 1   RETADR   RESW    1
100   0003 1   LENGTH   RESW    1
103   0000 2            USE     CBLKS
105   0000 2   BUFFER   RESB    4096
106   1000 2   BUFEND   EQU     *
107   1000     MAXLEN   EQU     BUFEND-BUFFER
110             .
```

0: default
1: CDATA
2: CBLKS

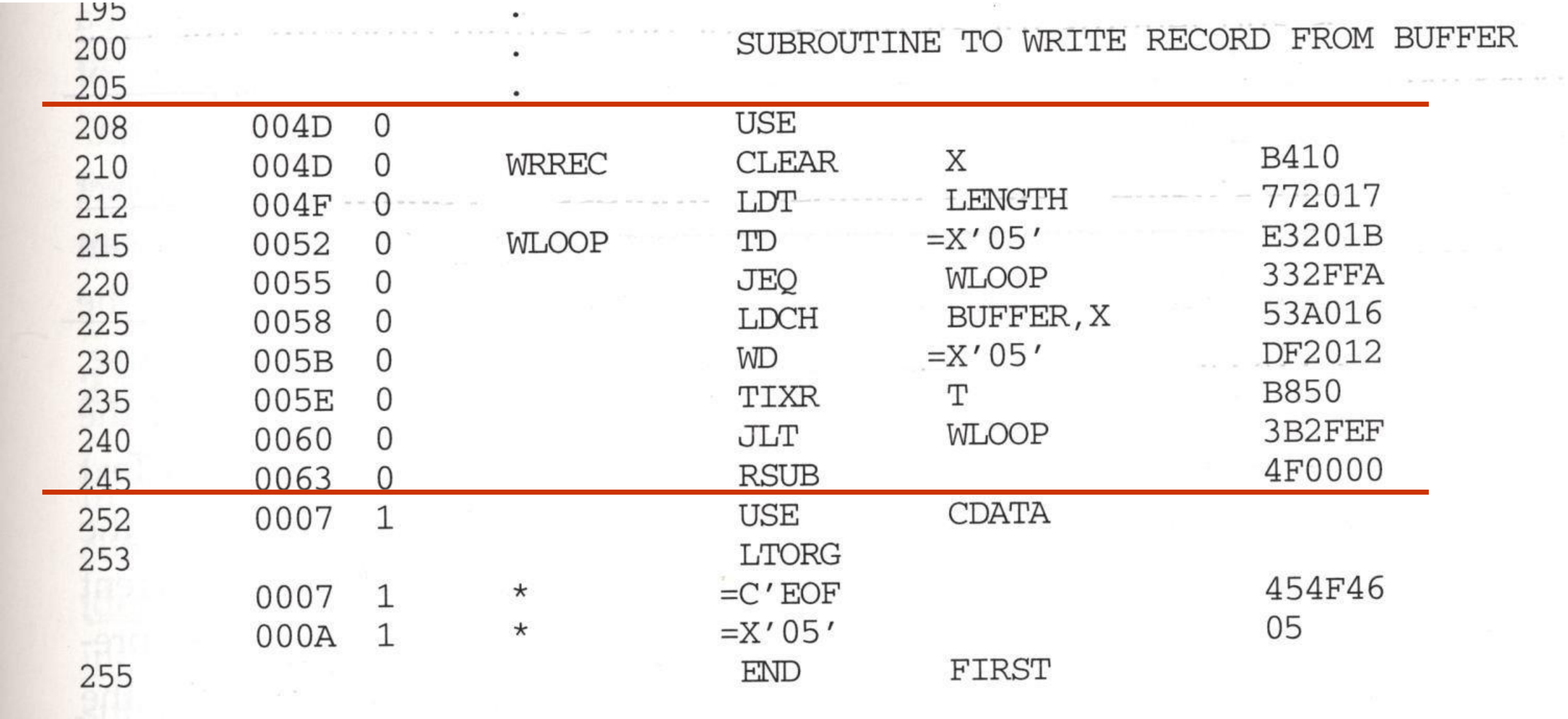
No block number because MAXLEN is an absolute symbol

Object Program with Multiple Program Blocks



```
110                 .
115                 .         SUBROUTINE TO READ RECORD INTO BUFFER
120                 .
123   0027  0                 USE
125   0027  0    RDREC        CLEAR     X                  B410
130   0029  0                 CLEAR     A                  B400
132   002B  0                 CLEAR     S                  B440
133   002D  0                +LDT       #MAXLEN            75101000
135   0031  0    RLOOP        TD        INPUT              E32038
140   0034  0                 JEQ       RLOOP              332FFA
145   0037  0                 RD        INPUT              DB2032
150   003A  0                 COMPR     A,S                A004
155   003C  0                 JEQ       EXIT               332008
160   003F  0                 STCH      BUFFER,X           57A02F
165   0042  0                 TIXR      T                  B850
170   0044  0                 JLT       RLOOP              3B2FEA
175   0047  0    EXIT         STX       LENGTH             13201F
180   004A  0                 RSUB                         4F0000
183   0006  1                 USE       CDATA
185   0006  1    INPUT        BYTE      X'F1'              F1
```

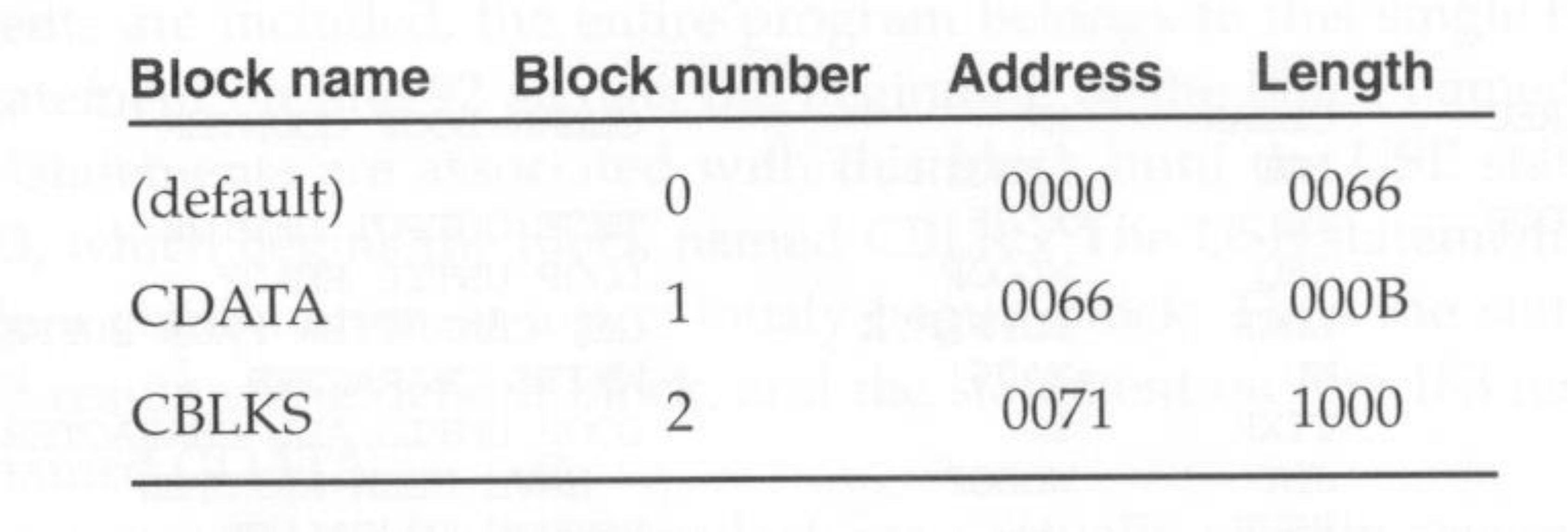
Object Program with Multiple Program Blocks



```
195                        .
200                        .          SUBROUTINE TO WRITE RECORD FROM BUFFER
205                        .
208      004D   0                     USE
210      004D   0          WRREC      CLEAR        X                 B410
212      004F   0                     LDT          LENGTH            772017
215      0052   0          WLOOP      TD          =X'05'             E3201B
220      0055   0                     JEQ          WLOOP             332FFA
225      0058   0                     LDCH         BUFFER,X          53A016
230      005B   0                     WD          =X'05'             DF2012
235      005E   0                     TIXR         T                 B850
240      0060   0                     JLT          WLOOP             3B2FEF
245      0063   0                     RSUB                           4F0000
252      0007   1                     USE          CDATA
253                                   LTORG
         0007   1          *         =C'EOF'                         454F46
         000A   1          *         =X'05'                          05
255                                   END          FIRST
```

Table for Program Blocks

- At the end of Pass 1:



Block name	Block number	Address	Length
(default)	0	0000	0066
CDATA	1	0066	000B
CBLKS	2	0071	1000

Example of Address Calculation

```
20      0006   0      LDA   LENGTH          032060
```

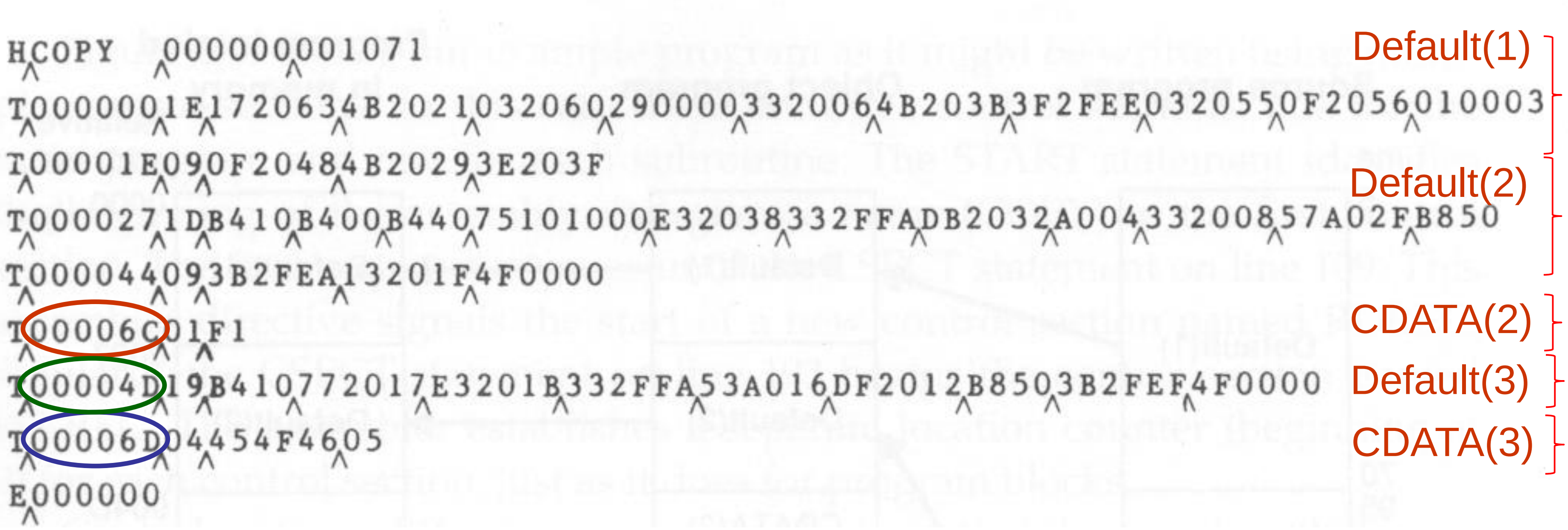
- The value of the operand (LENGTH)
  - Address 0003 relative to Block 1 (CDATA)

    →address 0003+0066=0069 relative to program

    →address 0069-0009=0060 relative to PC, in which the address of PC relative to program is 0009+0000=0009

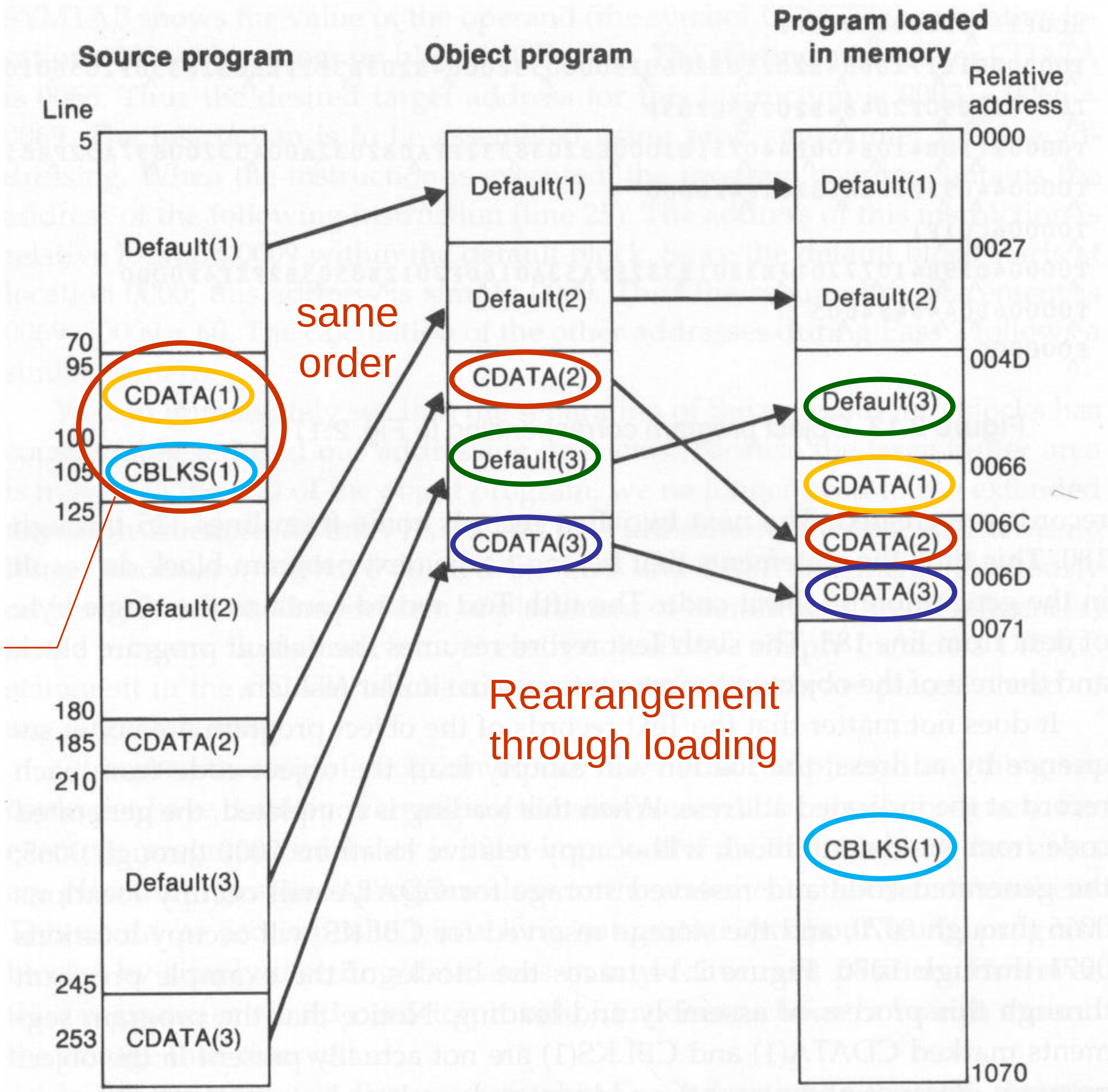
Object Program

- It is not necessary to physically rearrange the generated code in the object program to place the pieces of each program block together.
- The assembler just simply insert the proper load address in each Text record.



```
H^COPY  ^000000^001071
T^000000^1E^172063^4B2021^032060^290000^332006^4B203B^3F2FEE^032055^0F2056^010003
T^00001E^09^0F2048^4B2029^3E203F
T^000027^1D^B410^B400^B440^75101000^E32038^332FFA^DB2032^A004^332008^57A02F^B850
T^000044^09^3B2FEA^13201F^4F0000
T^00006C^01^F1
T^00004D^19^B410^772017^E3201B^332FFA^53A016^DF2012^B850^3B2FEF^4F0000
T^00006D^04^454F46^05
E^000000
```

Program Blocks Loaded in Memory



same order

Not present in object program

Rearrangement through loading

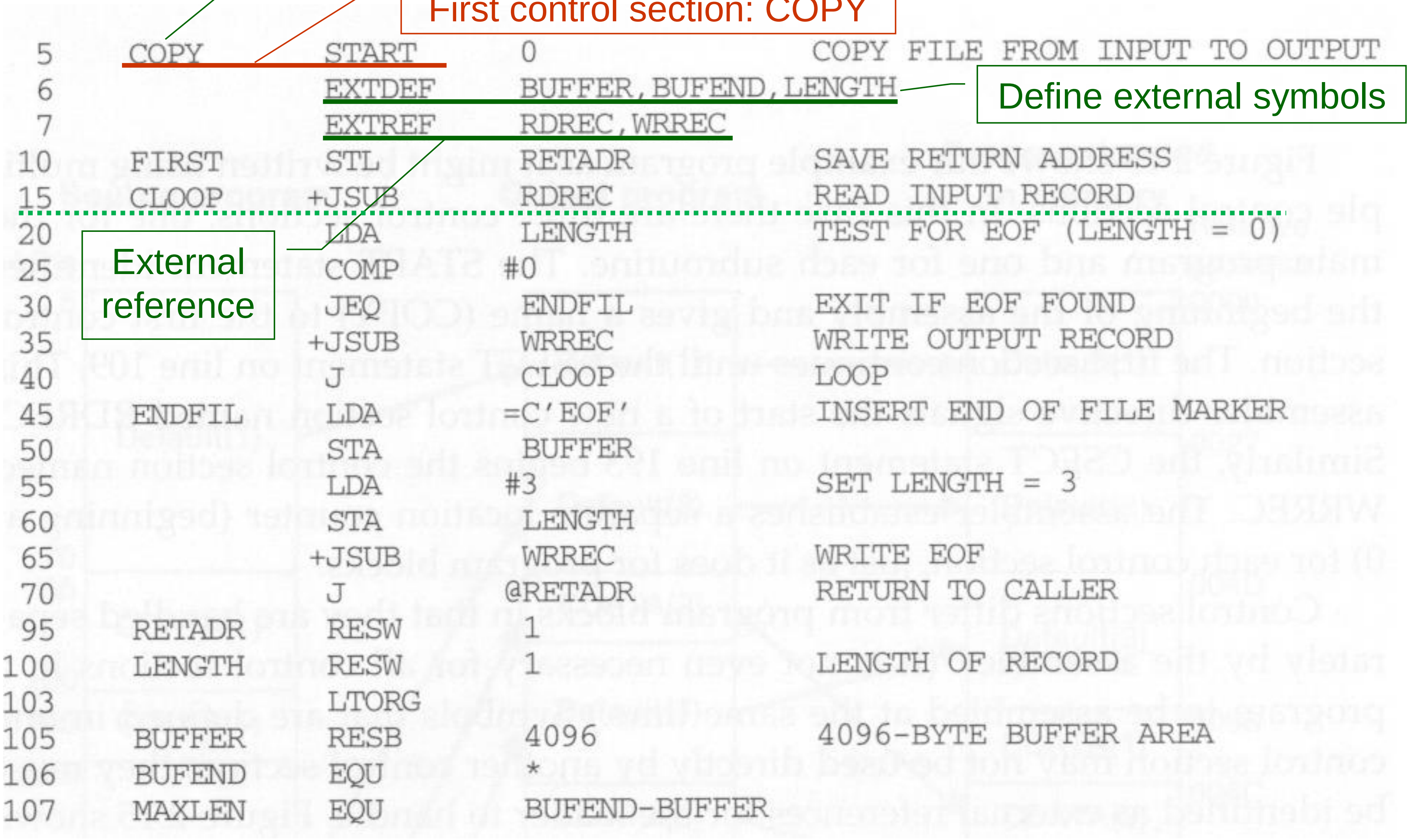
Control Sections

- A control section
  - is a part of the program that maintains its identity after assembly
  - is often used for subroutine or other logical subdivision of a program
  - can be assembled, loaded, and relocated independently
  - is more flexible

Program Linking

- Program linking is used to link together logically related control sections
- Problem:
  - The assembler does not know where any other control section will be located at execution time.
  - When an instruction needs to refer to instructions or data located in another control section, the assembler is unable to process this reference.
  - The assembler has to generate information for such kind of references, called external references, that will allow the loader to perform the required linking.

Program with Multiple Control Sections



```
  5     COPY      START     0               COPY FILE FROM INPUT TO OUTPUT
  6               EXTDEF    BUFFER,BUFEND,LENGTH
  7               EXTREF    RDREC,WRREC
 10     FIRST     STL       RETADR          SAVE RETURN ADDRESS
 15     CLOOP    +JSUB      RDREC           READ INPUT RECORD
 20               LDA       LENGTH          TEST FOR EOF (LENGTH = 0)
 25               COMP     #0
 30               JEQ       ENDFIL          EXIT IF EOF FOUND
 35              +JSUB      WRREC           WRITE OUTPUT RECORD
 40               J         CLOOP           LOOP
 45     ENDFIL    LDA      =C'EOF'          INSERT END OF FILE MARKER
 50               STA       BUFFER
 55               LDA      #3               SET LENGTH = 3
 60               STA       LENGTH
 65              +JSUB      WRREC           WRITE EOF
 70               J        @RETADR          RETURN TO CALLER
 95     RETADR    RESW      1
100     LENGTH    RESW      1               LENGTH OF RECORD
103               LTORG
105     BUFFER    RESB      4096            4096-BYTE BUFFER AREA
106     BUFEND    EQU       *
107     MAXLEN    EQU       BUFEND-BUFFER
```

Implicitly defined as an external symbol

First control section: COPY

Define external symbols

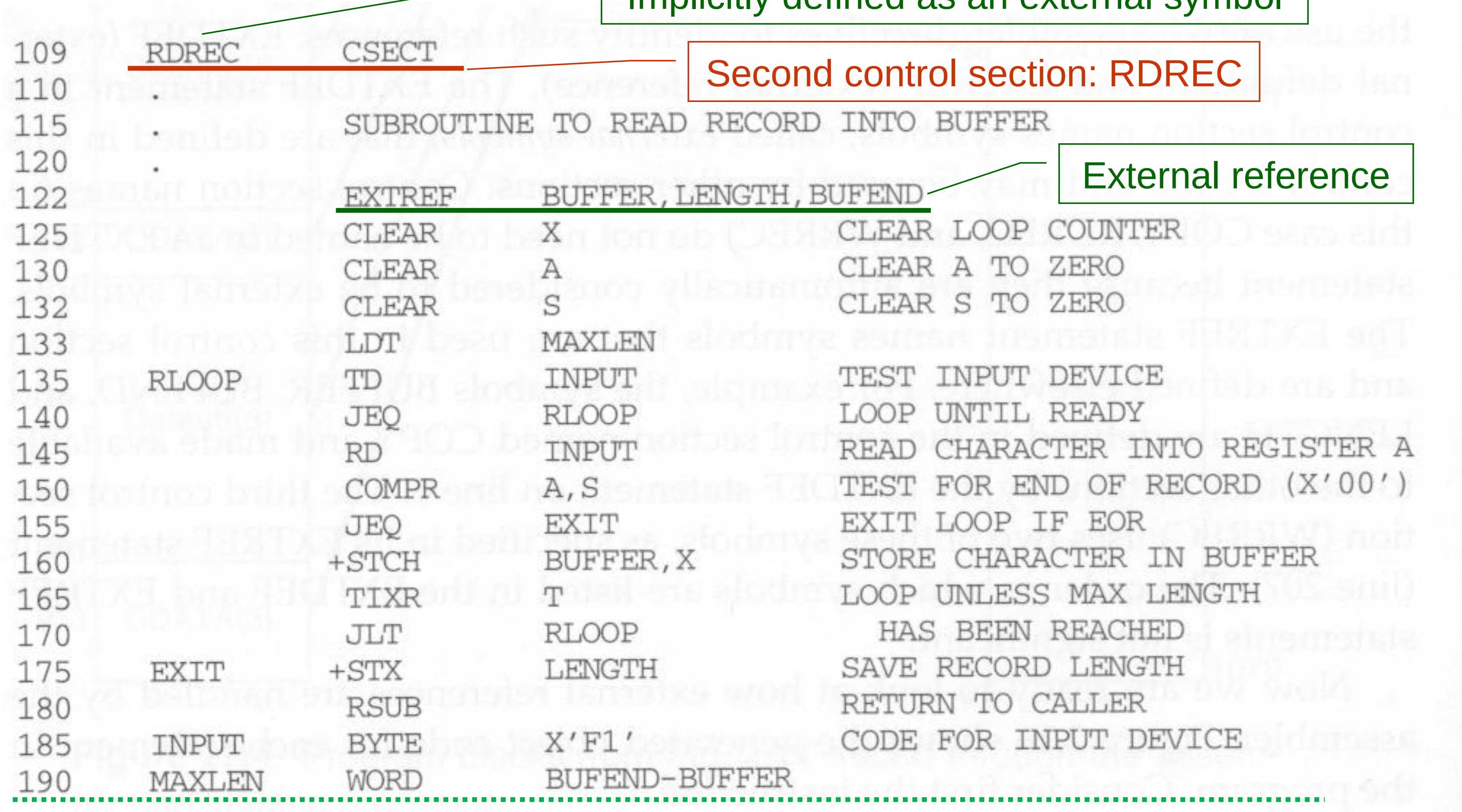
External reference

Program with Multiple Control Sections

Implicitly defined as an external symbol

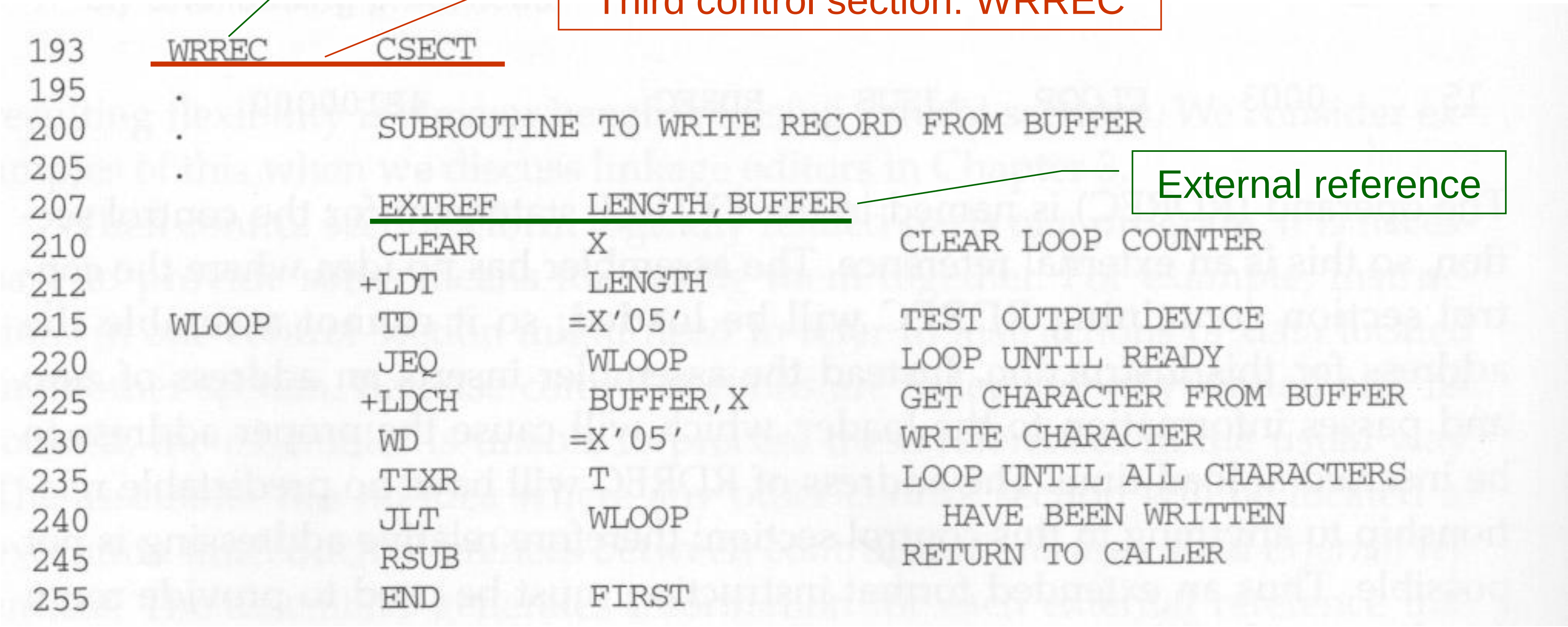
Second control section: RDREC

External reference



```
RDREC    CSECT
.
.        SUBROUTINE TO READ RECORD INTO BUFFER
.
         EXTREF   BUFFER,LENGTH,BUFEND
         CLEAR    X                CLEAR LOOP COUNTER
         CLEAR    A                CLEAR A TO ZERO
         CLEAR    S                CLEAR S TO ZERO
         LDT      MAXLEN
RLOOP    TD       INPUT            TEST INPUT DEVICE
         JEQ      RLOOP            LOOP UNTIL READY
         RD       INPUT            READ CHARACTER INTO REGISTER A
         COMPR    A,S              TEST FOR END OF RECORD (X'00')
         JEQ      EXIT             EXIT LOOP IF EOR
        +STCH     BUFFER,X         STORE CHARACTER IN BUFFER
         TIXR     T                LOOP UNLESS MAX LENGTH
         JLT      RLOOP              HAS BEEN REACHED
EXIT    +STX      LENGTH           SAVE RECORD LENGTH
         RSUB                      RETURN TO CALLER
INPUT    BYTE     X'F1'            CODE FOR INPUT DEVICE
MAXLEN   WORD     BUFEND-BUFFER
```

Program with Multiple Control Sections



```
WRREC   CSECT
.
.       SUBROUTINE TO WRITE RECORD FROM BUFFER
.
        EXTREF  LENGTH,BUFFER
        CLEAR   X               CLEAR LOOP COUNTER
       +LDT     LENGTH
WLOOP   TD     =X'05'           TEST OUTPUT DEVICE
        JEQ     WLOOP           LOOP UNTIL READY
       +LDCH    BUFFER,X        GET CHARACTER FROM BUFFER
        WD     =X'05'           WRITE CHARACTER
        TIXR    T               LOOP UNTIL ALL CHARACTERS
                                  HAVE BEEN WRITTEN
        JLT     WLOOP
        RSUB                    RETURN TO CALLER
        END     FIRST
```

Implicitly defined as an external symbol

Third control section: WRREC

External reference

Assembler Directives for Control Section

- START:
  - start the first control section
  - set program name as the control section name
  - define the control section name as an external symbol
- CSECT:
  - start a new control section
  - specify the control section name
  - define the control section name as an external symbol
- EXTDEF:
  - define external symbols
- EXTREF:
  - name symbols defined in other control sections

How to Handle External References

```
15    0003     CLOOP     +JSUB   RDREC      4B100000
```

- The operand RDREC is an external reference.
- The assembler
  - has no idea where RDREC is
  - inserts an address of zero
  - can only use extended format to provide enough room (that is, relative addressing for external reference is invalid)
  - passes information to the loader

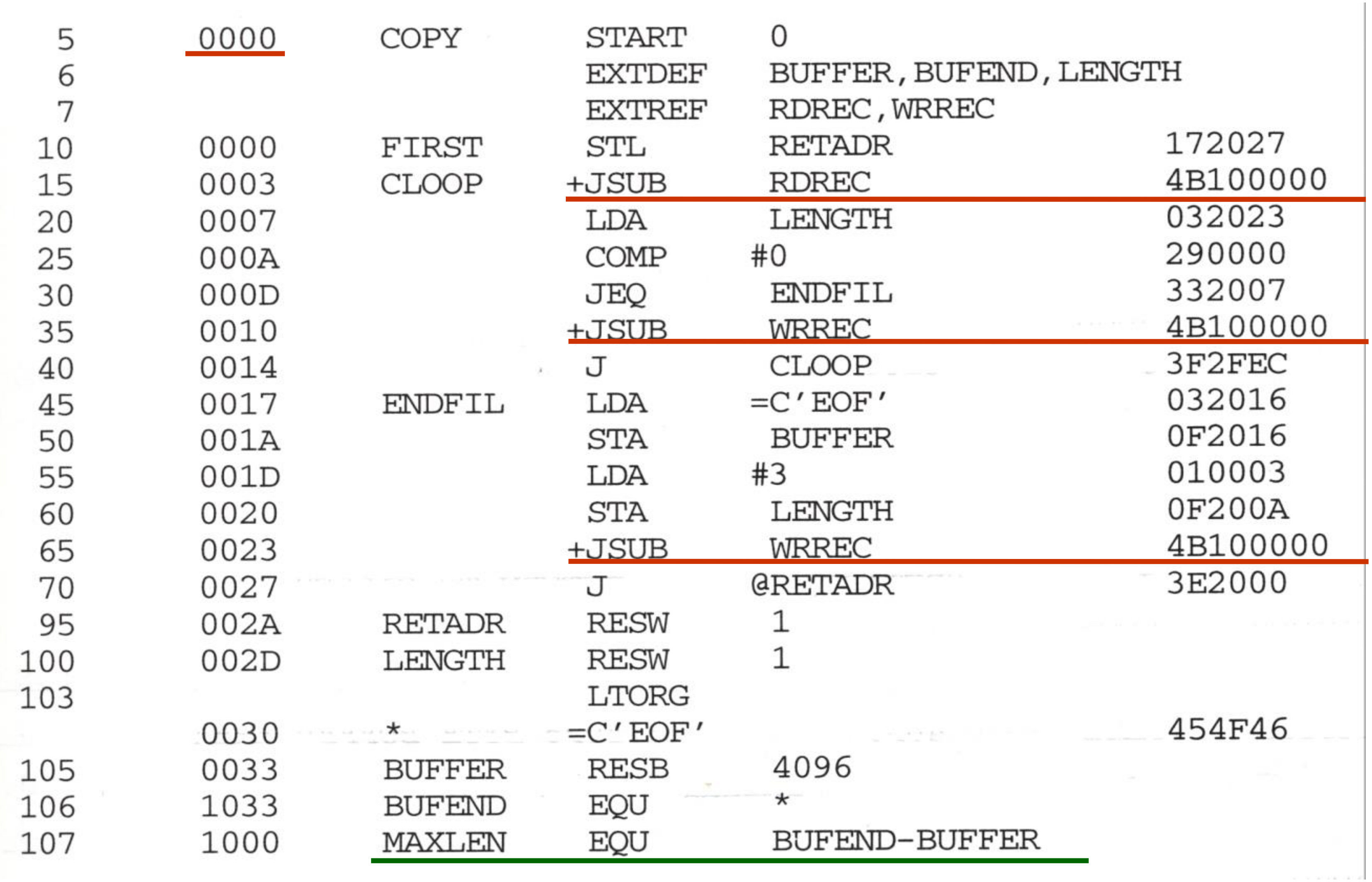
How to Handle External References

```
190   0028   MAXLEN   WORD   BUFEND-BUFFER   000000
```

- There are two external references in the expression, BUFEND and BUFFER.
- The assembler
  - inserts a value of zero
  - passes information to the loader
    - Add to this data area the address of BUFEND
    - Subtract from this data area the address of BUFFER
- On line 107, BUFEND and BUFFER are defined in the same control section and the expression can be calculated immediately.

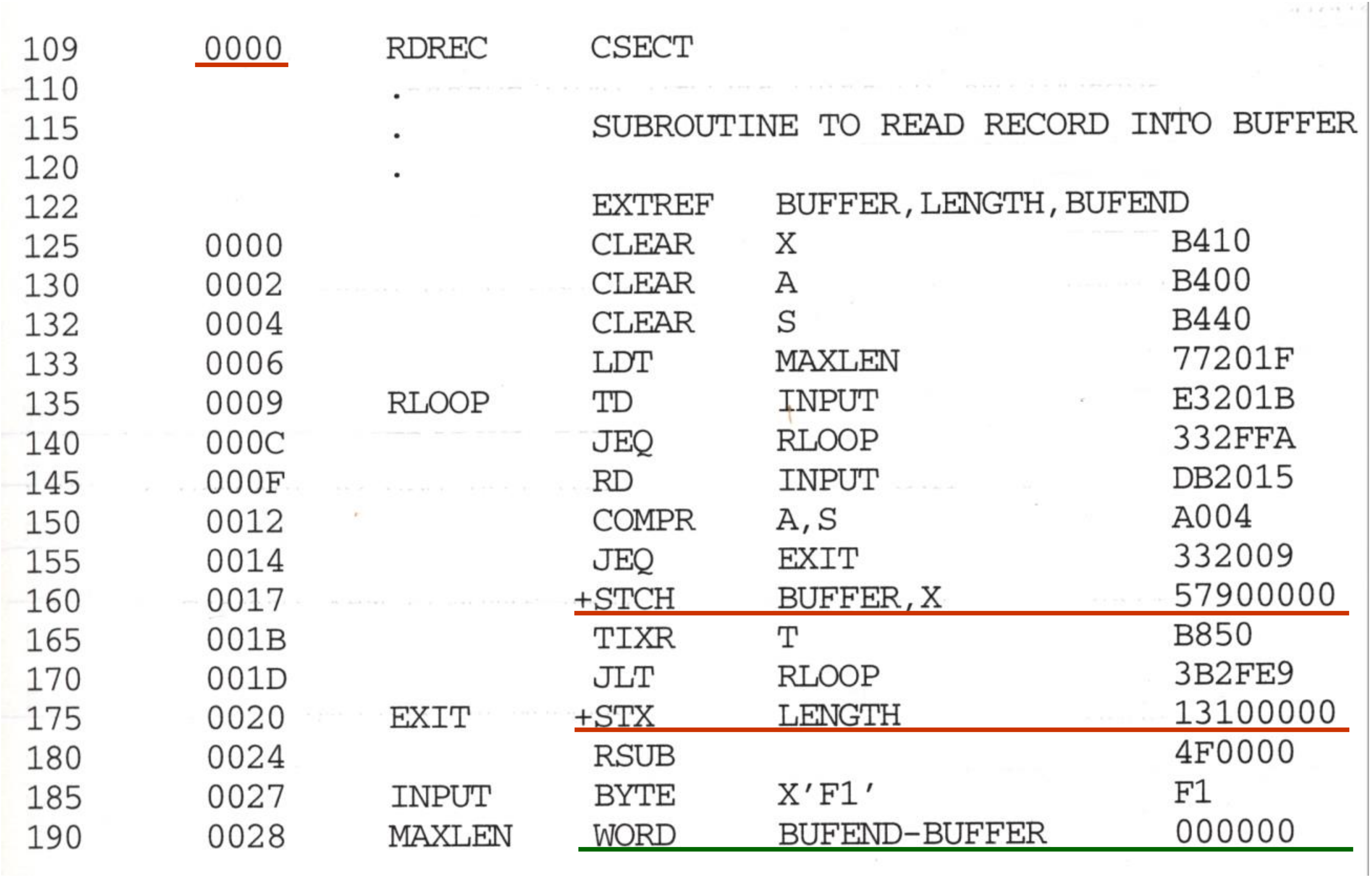
```
107   1000   MAXLEN   EQU   BUFEND-BUFFER
```

Object Code with Multiple Control Sections



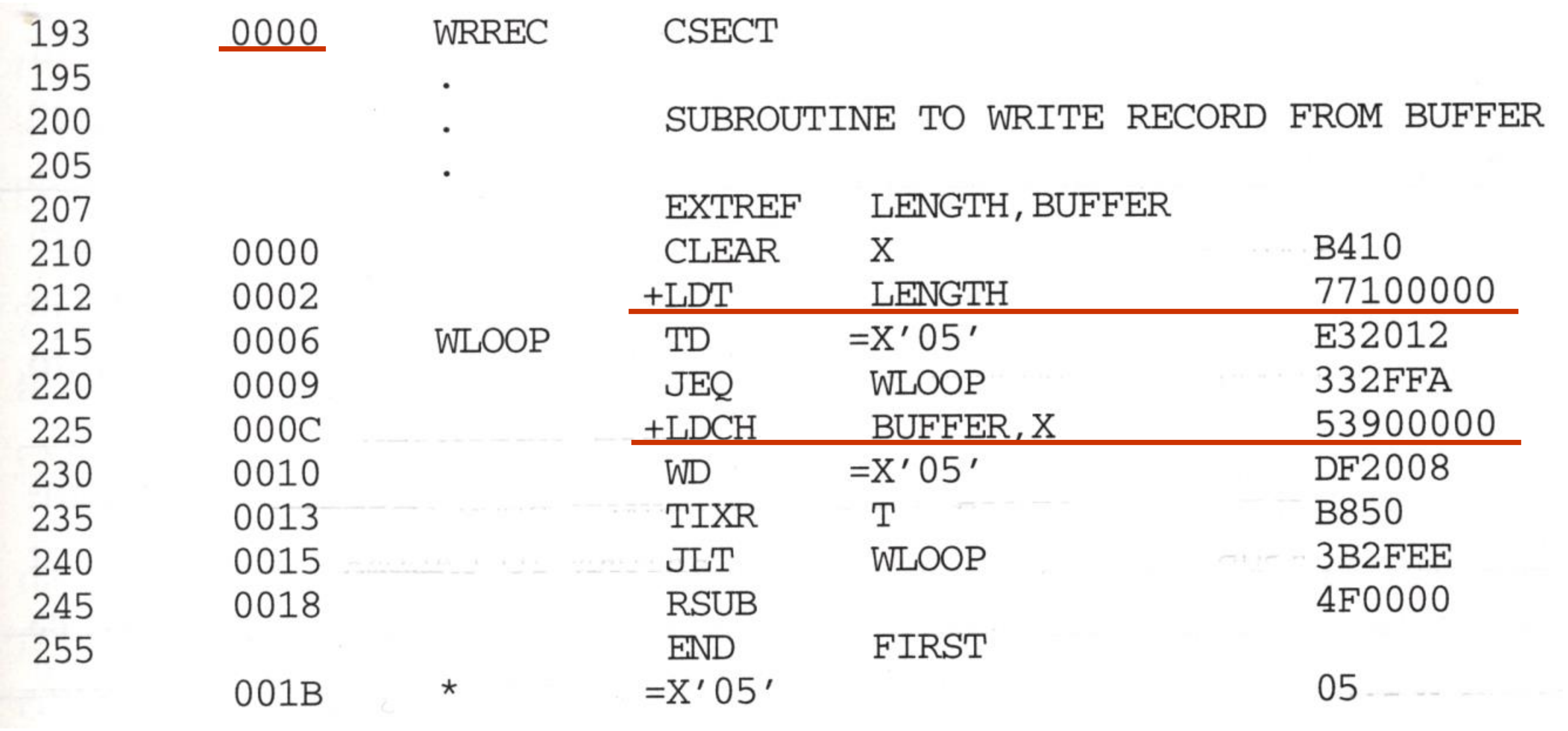
```
  5    0000    COPY      START     0
  6                      EXTDEF    BUFFER,BUFEND,LENGTH
  7                      EXTREF    RDREC,WRREC
 10    0000    FIRST     STL       RETADR               172027
 15    0003    CLOOP    +JSUB      RDREC                4B100000
 20    0007              LDA       LENGTH               032023
 25    000A              COMP     #0                    290000
 30    000D              JEQ       ENDFIL               332007
 35    0010             +JSUB      WRREC                4B100000
 40    0014              J         CLOOP                3F2FEC
 45    0017    ENDFIL    LDA      =C'EOF'               032016
 50    001A              STA       BUFFER               0F2016
 55    001D              LDA      #3                    010003
 60    0020              STA       LENGTH               0F200A
 65    0023             +JSUB      WRREC                4B100000
 70    0027              J        @RETADR               3E2000
 95    002A    RETADR    RESW      1
100    002D    LENGTH    RESW      1
103                      LTORG
       0030    *        =C'EOF'                         454F46
105    0033    BUFFER    RESB      4096
106    1033    BUFEND    EQU       *
107    1000    MAXLEN    EQU       BUFEND-BUFFER
```

Object Code with Multiple Control Sections



```
109    0000     RDREC    CSECT
110             .
115             .        SUBROUTINE TO READ RECORD INTO BUFFER
120             .
122                      EXTREF    BUFFER,LENGTH,BUFEND
125    0000              CLEAR     X                     B410
130    0002              CLEAR     A                     B400
132    0004              CLEAR     S                     B440
133    0006              LDT       MAXLEN                77201F
135    0009     RLOOP    TD        INPUT                 E3201B
140    000C              JEQ       RLOOP                 332FFA
145    000F              RD        INPUT                 DB2015
150    0012              COMPR     A,S                   A004
155    0014              JEQ       EXIT                  332009
160    0017             +STCH      BUFFER,X              57900000
165    001B              TIXR      T                     B850
170    001D              JLT       RLOOP                 3B2FE9
175    0020     EXIT    +STX       LENGTH                13100000
180    0024              RSUB                            4F0000
185    0027     INPUT    BYTE      X'F1'                 F1
190    0028     MAXLEN   WORD      BUFEND-BUFFER         000000
```

Object Code with Multiple Control Sections

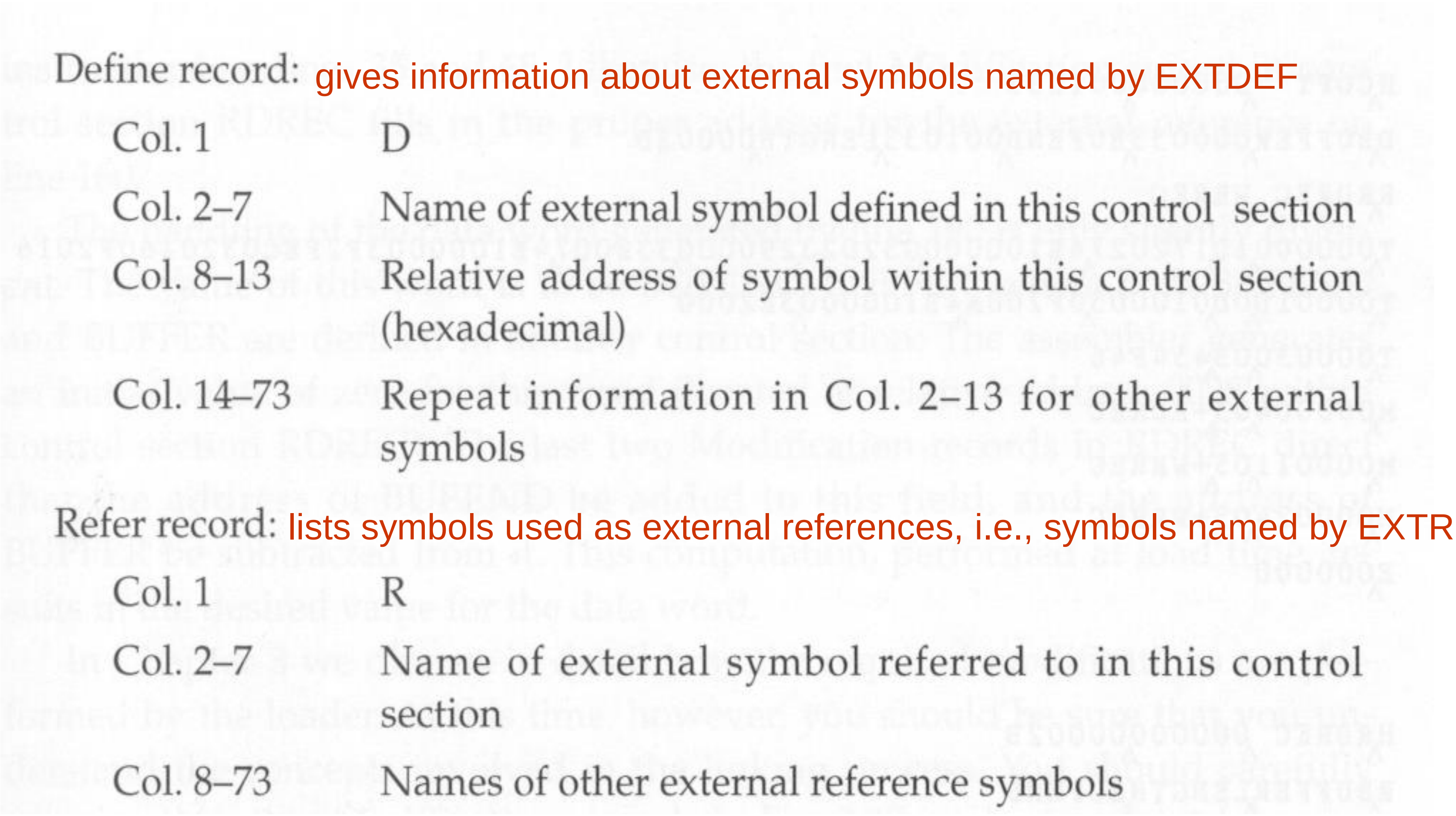


```
193     0000     WRREC     CSECT
195              .
200              .         SUBROUTINE TO WRITE RECORD FROM BUFFER
205              .
207                        EXTREF    LENGTH,BUFFER
210     0000               CLEAR     X                   B410
212     0002              +LDT       LENGTH              77100000
215     0006     WLOOP     TD       =X'05'               E32012
220     0009               JEQ       WLOOP               332FFA
225     000C              +LDCH      BUFFER,X            53900000
230     0010               WD       =X'05'               DF2008
235     0013               TIXR      T                   B850
240     0015               JLT       WLOOP               3B2FEE
245     0018               RSUB                          4F0000
255                        END       FIRST
        001B     *        =X'05'                         05
```

How to Handle Control Sections

- The assembler
  - processes each control section independently
  - establishes a separate LOCCTR (initialized to 0) for each control section
  - stores SYMTAB in the control section in which a symbol is defined
  - allow the same symbol to be used in different control sections
  - reports an error when attempting to refer to a symbol in another control section, unless the symbol is defined as an external reference
  - generates information in the object program for external references

New Records for External References



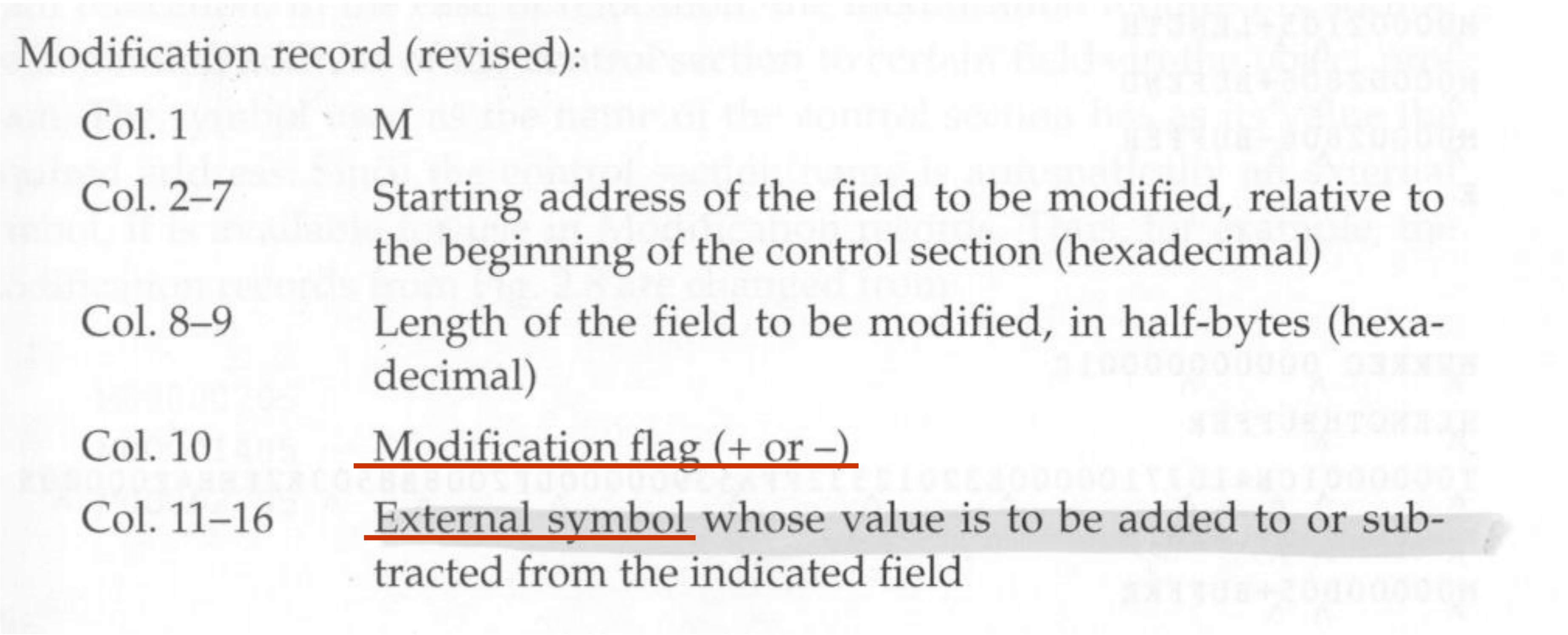
Define record: gives information about external symbols named by EXTDEF

Col. 1	D
Col. 2–7	Name of external symbol defined in this control section
Col. 8–13	Relative address of symbol within this control section (hexadecimal)
Col. 14–73	Repeat information in Col. 2–13 for other external symbols

Refer record: lists symbols used as external references, i.e., symbols named by EXTREF

Col. 1	R
Col. 2–7	Name of external symbol referred to in this control section
Col. 8–73	Names of other external reference symbols

Revised Modification Record



Modification record (revised):

Col. 1 M

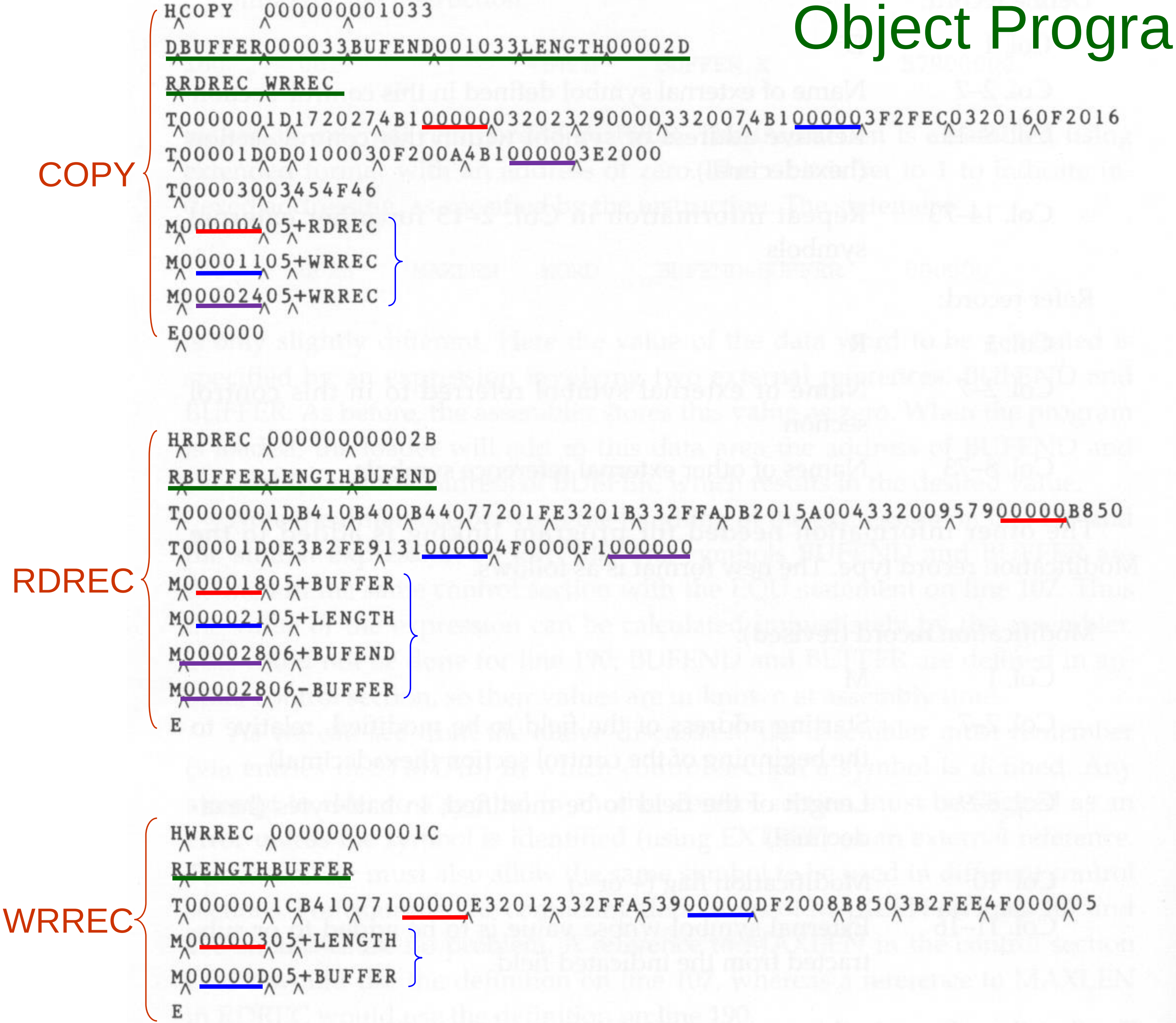
Col. 2–7 Starting address of the field to be modified, relative to the beginning of the control section (hexadecimal)

Col. 8–9 Length of the field to be modified, in half-bytes (hexadecimal)

Col. 10 Modification flag (+ or –)

Col. 11–16 External symbol whose value is to be added to or subtracted from the indicated field

Object Program



COPY

```
H^COPY  ^000000^001033
D^BUFFER^000033^BUFEND^001033^LENGTH^00002D
R^RDREC ^WRREC
T^000000^1D^172027^4B100000^032023^290000^332007^4B100000^3F2FEC^032016^0F2016
T^00001D^0D^010003^0F200A^4B100000^3E2000
T^000030^03^454F46
M^000004^05^+RDREC
M^000011^05^+WRREC
M^000024^05^+WRREC
E^000000
```

RDREC

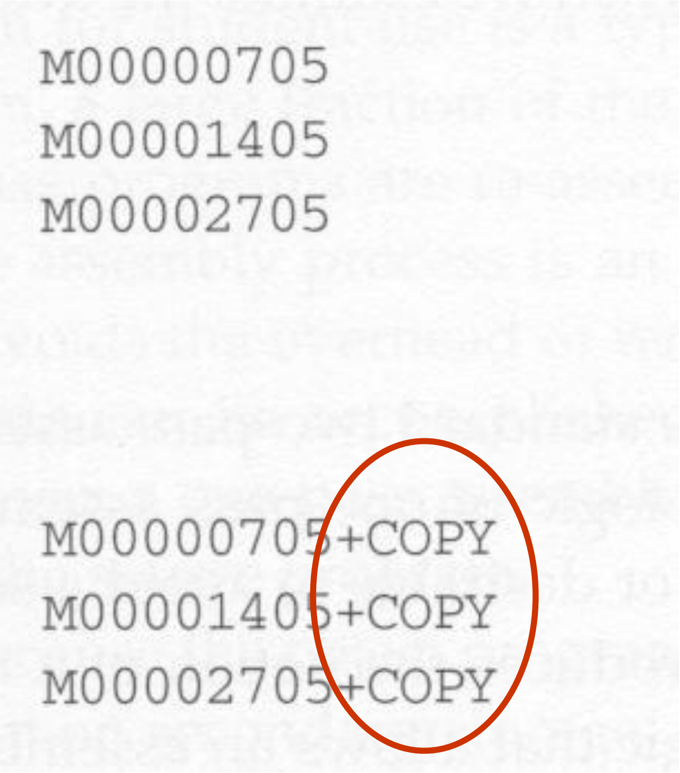
```
H^RDREC ^000000^00002B
R^BUFFER^LENGTH^BUFEND
T^000000^1D^B410^B400^B440^77201F^E3201B^332FFA^DB2015^A004^332009^57900000^B850
T^00001D^0E^3B2FE9^13100000^4F0000^F1^000000
M^000018^05^+BUFFER
M^000021^05^+LENGTH
M^000028^06^+BUFEND
M^000028^06^-BUFFER
E
```

WRREC

```
H^WRREC ^000000^00001C
R^LENGTH^BUFFER
T^000000^1C^B410^77100000^E32012^332FFA^53900000^DF2008^B850^3B2FEE^4F0000^05
M^000003^05^+LENGTH
M^00000D^05^+BUFFER
E
```

Program Relocation

- As well as for program linking, the revised Modification record may still be used to perform program relocation.



```
M00000705
M00001405
M00002705



M00000705+COPY
M00001405+COPY
M00002705+COPY
```

Expressions in Multiple Control Sections

- Extended restriction
  - Both terms in each pair of an expression must be within the same control section

    Legal: BUFEND-BUFFER

    Illegal: RDREC-COPY
- How to enforce this restriction
  - When an expression involves external references, the assembler cannot determine whether or not the expression is legal.
  - The assembler evaluates all of the terms it can, combines these to form an initial expression value, and generates Modification records.
  - The loader checks the expression for errors and finishes the evaluation.